1984

GPSS Compiler/Simulator

Jacquelyn Van Dellon

1. Preliminary Information.

1.1. Title and Acceptance Page.

Rochester Institute of Technology
School of Computer Science and Technology

Thesis:
GPSS Compiler/Simulator

by
Jacquelyn Van Dellon

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by : .....
(James Carbin - committee head)

.....
(Margaret Reek)

.....
(Guy Johnson)

Title of Thesis : **GPSS COMPILER/SIMULATOR**

I JACQUELYN VAN DELLON hereby grant permission to the Wallace Memorial Library, of RIT, to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

.....
(Jacquelyn Van Dellon)

9/7/84
DATE

1.2. Table of Contents.

1.3. Abstract.

As with any large software project, the first priority of this thesis is program correctness in the GPSS compiler and correct simulation statistics. Several distinct processes are gone through in the construction of any compiler/simulator: development of specifications, indication of the goals, design, implementation, evaluation, and maintenance.

The specifications of the GPSS compiler/simulator were introduced in the thesis proposal. The VAX/VMS 11/750 will be the machine which this GPSS compiler/simulator uses because of the supported 'C' language, a Command Language Definition Utility, and accessibility by this author. The implementation language, 'C', was chosen based on ease of readability, ease of understanding, the simple but powerful data structures in the language, and the language's support of modularization.

Three milestones were identified for this thesis project: the parser, the pseudo code generator, and the simulator. Many projects are started with only the end product in mind, but the goals which were identified constitute essential checkpoints to validate the integrity of the compiler/simulator.

There were three distinct design phases due to the milestone identification. The parser design encompassed the initial modular design layout of the source code for the entire compiler, and the design of the symbol table. Design of the pseudo code language had two constraints imposed: the ease of representation of the GPSS language in two-address code and the ease of integration of the pseudo language into the parser. Last but by no means unimportant was the design of the simulator, which controls the order of events within a model.

After each design phase described above was completed, the implementation or writing of the source code was accomplished. The parser, the pseudo code generator, and the simulator carried through the 'C' capabilities of modularization and readability.

Evaluation of the compiler was accomplished at three points: after the implementation of the parser, after the implementation of the pseudo code generator, and after the implementation of the simulator. The milestones or checkpoints specified in the identification of goals were essential in verifying the integrity of the system.

Although maintenance is not a part of this thesis, the design phase took into consideration the ease of adding functionality to the implemented subset of GPSS. This compiler/simulator could be used as a base from which a more sophisticated and current version of the GPSS language would immerge.

1.4. Key Words and Phrases.

BDT		*block departure time*
current events chain	-	the events chain a transaction is linked when block departure time equals the current clock time
discrete simulation language	-	a language which can represent a network of complex interactions, many of which are occurring at the same time
discrete function		a step function
first pass		builds symbol tables, creates skeleton pseudo code, and parses the program for language errors
future events chain	-	the events chain a transaction is linked when the transaction has an associated BDT (block departure time) greater than the current block time
GPDS	-	XEROX General Purpose Discrete System
GPSS	-	General Purpose Simulation System
one-pass compiler		fully generates the object code after one scan of the source
pseudo code	-	intermediate language which usually consists of a relatively small number of simple operations combined to form a list semantically equivalent to the source program [1]
scheduler	-	schedules routines/functions to run in a specified sequential order to produce simulation results
second pass	-	creates pseudo code for the scheduler
two-address code	-	intermediate language representation which consists of an operator, and two operands
two-pass compiler	-	fully generates the object code after two scans of the source

1.5. Computing Review Subject Codes.

Primary Code: D.3.4 Processors
Parser, Pseudo Code Generator, and Simulator

Secondary Codes: I.6 Simulation and Modeling
Concepts and understanding of the GPSS language and uses

E.2 Data Storage Representation
Binary tree structure for the symbol table
One way linked list for the pseudo code
Linked list for the future events chain

D.3.1 Formal Definitions and Theory
Break down of the GPSS subset into BNF representation
Requirements for implementing the GPSS compiler/simulator
Test criteria

2. Introduction and Background.

The following pages describe the process which was taken in the design and implementation of this GPSS compiler/simulator. Although the main consideration of this project was to deliver an efficient, well designed software package, the project's emphasis is on the pseudo code generator and its integration into the parser.

Geoffrey Gordon, an employee of IBM, developed the General Purpose System Simulator, GPSS, and presented the concept in two papers: "A General Purpose Systems Simulator" in 1961, and "A General Purpose Digital Simulator and Examples of its Applications: Part I - Description of the Simulator" in 1962. The IBM 704, 709, and 7090 computers were the machines on which the first release of the GPSS language, often called the "Gordon Simulator", was implemented.

Systems engineers of the early 60's had come to realize simulation was a valuable tool which could be used to analyze the performance of proposed systems configurations. Simulation, properly used, could provide valuable information by showing in some detail how the components of a system were expected to behave.[2]

The problem of preparing a computer simulation in the early 1960's fell into two tasks: first, a model of the system had to be constructed and second, a computer program had to be written to effectively "run" the structured model. The process was simplified by the introduction of the GPSS language. No longer were long programs required for each model. All the engineer had to do was to describe his problem in terms of the GPSS language and then let the GPSS simulator model the system.

The language was not designed for any particular class of systems. The aim was to define certain basic actions which were characteristic of systems so that the program could be applied to a wide variety of different systems applications.[3] The language was targeted for systems engineers who were not specialists in computer programming. The use of flowcharts to describe a system was well known at this time, and so GPSS was structured as a block-oriented language. This philosophy allowed the analyst to describe the system as a connected network of block diagrams representing the sequence of events.

A set of 25 specific block types was defined for the first implementation of GPSS. Each block represented a basic system action and had an associated time to perform this action. The system was described in terms of a combination of blocks which could be used repeatedly. General units of traffic in the system were represented by transactions, which were defined to model the dynamic components of a system. The transaction or basic unit of GPSS could vary in nature, depending on the system. For example, in a traffic model the units might be cars, or in a grocery store simulation the units might be people, or in a communication system the units might be packets. The transactions moved through the simulation under the control of the blocks and were created and destroyed as required.[4]

The fundamental concepts of block-diagram structure and transactions were retained in GPSS II, released in 1964, and all functions performed by the earlier program could also be performed by GPSS II. A summary of improvements were:

* A greater ability to sense the current state of the system, and to implement decisions based upon this state. "VARIABLE statements" permitted FORTRAN-like algebraic computations using system variables. Those capabilities provided improved control over the flow of transactions in response to the current state of the system.

* The ability to associate a greater amount of information with each transaction in the form of eight parameters.

* The introduction of an indirect specification feature which permitted a transaction to specify from its parameters the characteristics of the block entered, rather than requiring these characteristics be fixed for an entire simulation. This feature added flexibility and made it possible to reduce the size of certain types of models.

* The generalization of GPSS functions to permit a wider variety of arguments and a greater number of data points for inserting data descriptive of the system.

* An optional assembly feature which simplified the description of the block diagram by furnishing the block numbers from symbolic names, and which enabled the program to set up and call "block macros". These macros were user-defined segments of a block diagram which could be used repetitively within a model, with the block characteristics varied as desired.

* The ability to go into FAP (FORTRAN application program) subroutines prepared by the user.

* Expanded output statistics and error information.

* Generally faster execution. 5

GPSS III, released in 1965, was extended for use on the IBM 7040 and 7044. Several distinct advantages, brought about by a major language design, were seen in this new version. Time delays, which were previously incorporated into every block, were eliminated and replaced by a single block, called an ADVANCE. Selection factors were also removed from all blocks and replaced by the TRANSFER block. These changes redefined the whole block structure but made the conceptual understanding much easier. A summary of other changes to GPSS III are listed below.

* Increased the speed of the simulator due to the language redesign.

* Gave the user the capability of choosing a variable number of parameters per transaction.

* Storage flexibility of automatic space reallocation, gave the user the ability to reapportion core storage without going through a reassembly phase.

* The introduction of user chains, allowed users to create their own chains independent of the standard Current Events, Future Events, or Interrupt chains.

* The new Attribute Valued function allowed the use of any Standard Numerical Attribute in the specification of a function point where previously only constants could appear.

* Increased debugging tools. The START card had the snapshot feature added, which printed out statistics. The new block, PRINT, could print any Standard Numerical Attribute.

* The addition of the DEPART block, split the function of the QUEUE block into two separate blocks. This was done in order to get better statistics from the QUEUE block.

* The addition of two new features to the SPLIT block: the ability to specify the number of transactions to be split, and the ability to serially copy transactions in a specified parameter . GATHER, a new block, removes transactions from the Current Events chain until the count specified is reached.

* The HELP block was extended to include symbolic addressing.

* The addition of a new function, List, required the independent argument values of the function be sequential integers beginning with 1. This feature allowed the function to be evaluated more quickly than a comparable discrete function and, since arguments were not stored, also saved core.

* The addition of the subroutine mode to the TRANSFER block allowed specified subroutines to be called and returned the user to the proper block.

* The addition of two new blocks, EXECUTE and CHANGE, increased the GPSS flexibility. The EXECUTE block allowed the entering transaction to perform the operation of any other specified block without diverting the transaction from its normal sequential flow. The CHANGE block allowed the user to modify the model during the course of a simulation by changing any one block to a duplicate of any other block.

The basic structure of the language has remained unaltered since GPSS III and all further releases are upward-compatible.[8]

GPSS/360, announced in 1967, was more versatile because it could operate under IBM's OS/360 and DOS/360. The major changes to the language were:

* The addition of signed halfword and fullword parameters and savevalues increased the range of the quantity in a parameter. Prior to GPSS/360 parameters were considered to be positive.

* Additional Standard Numerical Attributes were added increasing the versatility of the simulator. The new Standard Numerical Attributes were: the priority of a transaction, the utilization and number of entries for facilities and storage, and other statistics relating to table, facilities, queues, and user chains.

* The new blocks, COUNT and SELECT, provided the user with the means to status multiple entities. The COUNT block determines the number of entities that meet a specified condition within a range. The SELECT block determines the number of an entity that meets a specified condition within a range.

* The group entity allowed the user to associate transactions based on like attributes.

* Matrix savevalues allowed the user to access and define a two-dimensional array of main storage locations.

* The operation of the PREEMPT now takes into consideration the priority of the transaction; GPSS/360 allows a transaction to have a priority from 1-127.

* Eight random number generators gave the users independent sources of random numbers. Parametric studies often require that specific delays based on a random number in certain paths of a model be identical from run to run. The user now has the ability to reference unique random-number generators in such paths and can be assured that changes in other parts of the model will not affect the consistent progress of transactions in the desired paths.[9]

* Boolean variables were introduced, thus increasing the logical power of the language.

* Macro instructions were introduced, thus helping the modeler reduce redundant code.

* The HELP block was introduced allowing the analyst an interface to assembly language programs.

* An output editor and the capability of producing histograms was supplied allowing the analyst a departure from the standard output. The new capabilities helped to extend the outputs readability and give a more meaningful report for later use.

Two additional releases have since been made, a second version of GPSS/360 and GPSS V. These changes were directed mainly to simplify the routine tasks of modeling and to relax restrictions on the size of the models. A run timer can be set to limit the total computer execution time used. When run time expires before the end of simulation, output is produced and the simulation run is terminated. The HELP block has been extended so that the analyst has a simple interface to the power of FORTRAN or PL/I. Free-form coding reduces the labor of program preparation. New types of parameters and arrays give more flexible data handling within the model, and the number of parameters that can be associated with each transaction has been increased significantly.[10]

The history, given above, shows a logical progression of the GPSS language development over the years. The choice of a GPSS subset for this thesis implementation was based on the history of the language and the needs of the user. Three questions needed to be answered: 1. How many GPSS instructions were implemented for the initial release of the language? 2. Since only the standardized report was to be allowed, what instructions had to be included to gather the specific statistics? 3. What GPSS instructions were needed to give the modeler flexibility?

Twenty-five instructions made up the first version of GPSS. The assumption was that approximately the same number of instructions would give the analyst using the thesis GPSS version reasonable modeling capabilities. Three divisions of the GPSS language, Control Statements, Entity Definition Statements, and Block Definition Statements, were examined, and a selection process was undertaken. The following paragraphs describe the subset chosen, where the GPSS subset roughly corresponds to an implementation of GPSS III.

Within the Control Statements, the **END**, **SIMULATE**, and **START** statements were chosen. The **END** and **SIMULATE** were chosen for compatibility to previous versions of the language. The input medium in the early 1960's was computer cards and the **END** statement served as the termination card for the program. Since file processing is the chosen medium for this thesis, clearly this instruction is not required. The **SIMULATE** statement signifies that the program should be parsed and simulated; if the statement was not included in the program only the parsing phase would occur. The command line could include the required qualifier to perform the **SIMULATE** instruction. The **START** statement marked the beginning of the simulation phase and specified the run termination count when the model should end. The thesis **START** card specifies the run termination count, an optional switch for output, and an optional operand which specifies snap interval printout of statistics.

Entity Definition Statements define permanent entities of the model. The **FUNCTION** statement was included because it allows the modeler to specify data as x and *y* coordinates. Only the discrete function was implemented for the thesis; GPSS V has the capability of five functions continuous, discrete, discrete attribute valued, list, and list attribute valued. The **VARIABLE** statement allows for arithmetic computations within the model framework. The **STORAGE** statement was included as one of the required statements for the standardized output.

The Block Definition Statements form the logical network of the model. The **GENERATE** and **TERMINATE** are the most important GPSS instructions. The **GENERATE** creates the transaction which enters the model and the **TERMINATE** destroys the transaction. The **ADVANCE** block gives the analyst the capability of modifying the

simulation clock; delaying a transaction for a specified amount of time. The ASSIGN instruction allows the modeler to arithmetically modify one of the ten parameters which are logically connected to the transaction. The GATE, TRANSFER, and TEST instructions were included to give the ability of moving in a nonsequential mode through the model. The QUEUE, DEPART, ENTER, LEAVE, SEIZE, and RELEASE were all instructions needed to generate the standard statistical output.

Listed below is a summary of the chosen subset of GPSS statements, and the respective implemented version of GPSS.

	GPSS statement	GPSS implemented version
CONTROL STATEMENT:		
	END	GPSS I
	SIMULATE	GPSS I
	START	GPSS III
ENTITY DEFINITION STATEMENT:		
	FUNCTION	GPSS I
	STORAGE	GPSS I
	VARIABLE	GPSS II
BLOCK DEFINITION STATEMENT:		
	ADVANCE	GPSS III
	ASSIGN	GPSS III
	DEPART	GPSS III
	ENTER	GPSS III
	GATE	GPSS III
	GENERATE	GPSS III
	LEAVE	GPSS III
	QUEUE	GPSS III
	RELEASE	GPSS III
	SEIZE	GPSS III
	TERMINATE	GPSS III
	TEST	GPSS III
	TRANSFER	GPSS III

The following paragraphs describe further rules and restrictions applied to the GPSS subset of nineteen statements chosen for the thesis implementation:

Each transaction has ten parameters. The GPSS programmer does not have the capability to specify the number of parameters per transaction, an option introduced in GPSS III. Each parameter is a thirty-two bit integer, with a range from 2,147,483,647 to -2,147,483,648.

Four Standard Numerical Attributes are allowed within a block statement of a GPSS program: the function, the variable, the parameter, and the storage.

Free form input has been implemented which was first seen in GPSS V. The impact to the programmer is there are no column restrictions.

No user formatting of the simulation statistical output is allowed, the same constraints as GPSS III.

3. Functional Specification.

The purpose of this thesis is to take a GPSS program using the subset of nineteen block statements described in section 2. *Introduction and Background* on page 9, and parse the program for grammatical errors, compile the program, and output a standard statistical report.

The parsing stage of the compiler will parse from left to right checking for syntax errors per the BNF specification of the GPSS language found in *Appendix A*, page 187. Errors are flagged with an '*' and the appropriate error message is output if the label definition is incorrect, if the keyword definition is incorrect, if the mnemonic specification of a comparison is not valid, if the specification of the operands is incorrect, if the correct standard numerical attributes are not specified correctly per the keyword found on the line, and if extraneous information is found at the end of the instruction.

The compilation of the program consists of converting the GPSS program into a simulation usable form; an intermediate coding stage. This pseudo code generation has to follow the mapping algorithm shown on page 20.

The pseudo code generated, from the compilation phase, is used to form the GPSS chains. GPSS chains, both the Current and Future Events Chains, are lists of events occurring in the model currently and in the future. The simulation phase gathers statistics on the chains and then outputs the standard statistical report.

4. Architectural Design.

4.1. Parser and Symbol Table Design.

The first step, in designing the parser, was to break down the language into its basic components, in order to aid in the decision of utilities and the design of essential modules of code. A top down design approach was chosen, therefore the User's Manual for the GPSS language was written. Then to facilitate the breakdown of the grammar into units, the GPSS language was assumed to contain no ambiguity. The Backus-Naur notation was chosen to represent the definition of the language syntax. *Appendix A* on page 187 contains the BNF representation which was created for this thesis.

Using the syntactic definitions in *Appendix A*, the software utilities implemented were: parsing for a correct label, parsing for an integer number, parsing for a floating point number, parsing for an implemented standard numerical attribute, parsing for statement keywords, getting a word, processing errors, and building a symbol table.

The compiler/simulator was designed in three segments, therefore the software package is modularized. There is a controlling, main routine which monitors the running of pass one (the parser and the skeleton pseudo code generator), pass two (finish pseudo code creation), and the running of the simulator.

Pass one has a controlling, main routine which gets a line of source code, and passes control to the correct parsing function for the statement. The parsing of the keyword is also implemented in modules, therefore allowing simple, quick additions of statements, and options.

The design of the symbol table had three phases, understanding the purpose of the symbol table, understanding what information needed to be in the symbol table, and the method of accessing the data.

There are three purposes for a symbol table, listed in the literature read: to relate a name (variable) to where it is used in the program, to define the data type, and to make the internal program more concise by placing an indicator in the internal program pointing to the symbol table entry.[6] The thesis symbol table defines the data attribute, contains all attribute information assigned to the symbol, for example **FUNCTION** *x* and *y* data points, and makes the intermediate code more concise because the operands for each pseudo code instruction are pointers to the symbol table definitions.

There are five capabilities a symbol table should have per the literature read: to determine whether a given name is in the symbol table, to add a new name to the table, to access information in the table, to add new information in the table for a given name, and to delete a name or group of names from a given table.[7] The thesis program tool/function which deals with the symbol table can add new symbols to the table and can search the table for a specific symbol, to determine if the symbol has been defined or to access needed information.

The binary tree structure was chosen for the accessing method into the symbol table. GPSS is a simply structured language and the retrieval of information from the table was not viewed as a bottleneck of the parser or the pseudo code generator. Thus, the increased speed of the hash approach and the complexity of implementing such a scheme was not justified.

The implementation language of C strongly influenced the choice of the binary tree structure due to the ease of link representation. The following describes the main structure of the binary tree:

```
struct SYM-TABLE
    {
    char                *label ;
    struct LAB          *attr_pnt ;
    struct SYM-TABLE *right ;
    struct SYM-TABLE *left ;
    }
```

The links to the lower levels of the binary tree, **right** and **left**, are represented as pointers. The pointers make effective use of the compilers memory space. The **label** variable is a pointer to the ASCII representation of the symbol again making effective use of memory space. The label is the search field through the binary tree. The **attr_pnt** points to the associated label attributes.

The label attribute structure, **LAB**, was designed to retain the GPSS context for each label contained in the tree. Again, the **C** language strongly influenced the structure created for retaining tnis information.

```
struct LAB
    {
    int     lab_attribute ;
    int     leng_info ;
    int     *pnt_info ;
    int     func_spec ;
    double *flo_num ;
    }
```

lab_attribute defines the label attribute: STORAGE, LABEL, VARIABLE, FUNCTION, PARAMETER, TEMPORARY VARIABLE, or CONSTANT. The actual label information is pointed to by **pnt_info** and the length of that information is specified by **leng_info**. **func spec** defines the type of function. Although, the discrete function only is implemented in this version of GPSS, the variable was included for expansion purposes. The *y* data of a **FUNCTION** statement is contained in the **flo_num**. The **C** structure allows effective use of the compilers memory management space.

4.2. Pseudo Code Generator.

Intermediate languages can be in many forms: postfix, prefix, triples, and quadruples. Most intermediate languages are devised for code optimization. In designing the pseudo code generator, triples and quadruples were investigated.

Triples, or two-address code, contain three fields: an operator, and two operands. The results of the triple operation is referred to in the program by the number of the two-address code statement. The concept of indirect triples is a list of pointers to the actual two-address code statements. When pseudo code is being optimized, indirect triples allow a quick way to change the order of the code with out having to relocate the actual triples.

Quadruples, three-address code, on the other hand have: an operator and three operands. The three operands are the two variables which are being operated on and the last operand is for storing the result of the operation. Three-address code results may be program variable or program generated temporary locations.

Operations which can be represented by quadruples and triples are: math and arithmetic functions, assignments, indexing, comparisons, logical operations, branches, conditional and unconditional, begin or end block, and allocation of array storage.

In general a compiler is described as being broken up into three phases: the lexical pass, syntactic analysis, and code generation. The lexical pass phase scans an input string, detects markers and key words, and reduces the input string. The syntactic analyzer takes the inputted lexical string and outputs a tree structure division of verb, subject, and completes the symbol table. Code generation converts the syntactic analysis to a local form. It is common to combine the lexical and the syntactic analysis pass.

The two-address code or triples representation was the choice for the intermediate language for this thesis. One of the major advantages of this selection was the wide range of operations which could be produced by using triples. Not only was this suitable for the subset of the language implemented but this allows for the expansion of the GPSS options at a later date. Another reason the language was suited to the two-address code structure was the GPSS operand specifications for each block statement; a minimum of no operands to a maximum of three operands per statement. Please refer to chapter 14, the *User Manual*, for an in-depth look at the syntax for the block statements.

After reading research materials which described the IBM OS/360 H FORTRAN IV compiler and its use of the second phase to create quadruples (three-address format) and Control Data Corporation's FORTRAN extended compiler use of pass one to create a symbol table while generating an intermediate form of code, the decision was made to redesign the first pass of the compiler, the parser. It was clear, because of the examples of the IBM and Control Data Corporation's FORTRAN compiler, several operations could be combined into one pass of the code. The skeleton of the GPSS pseudo code could be generated in the first pass of the compiler, therefore optimizing the number of passes being made through the program. The second pass would be used only to complete the pseudo code structure. The label addresses of the TEST, TRANSFER, and GATE block were the only block statements which required completion after the first pass of the compiler.

Although the code is not used for optimization purposes in this version of GPSS, future implementations could utili_e this facility.

The implementation language of C strongly influenced the choice of the intermediate code representation. The C structure designed to represent the pseudo code is shown below:

```
struct  CODE
    {
    int             block
    int             operator
    SYM-TABLE       *A_operand
    SYM-TABLE       *B_operand
    struct CODE     *code_link
    };
```

The pseudo code structure is an effective use of the compilers memory management space because of the use of the operand pointers into the symbol table. **Operator** is an integer representation of one of the thirteen block statements. The **A_operand** and **B_operand** are pointers to the symbol table representations of the parameter, variable, function, storage, or constant used in the block statement. This tightly linked relation between the symbol table and the pseudo code representation avoids redundancy of information stored. The **block** is an integer between 0 and the total number of blocks and is used for offset purposes. The **code_link** is a pointer to the next pseudo code structure.

The list below describes the mapping algorithm from the block definition statements into two-address code.

Statement	Operator	A-operand	B-operand	Comment
ADVANCE	advance	A-operand	NULL	
ASSIGN	*neg*	*A-operand*	*B-operand*	*if B-operand minus*
	assign	A-operand	B-operand	
DEPART	depart	A-operand	B-operand	
ENTER	enter	A-operand	B-operand	
GENERATE	generate	A-operand	B-operand	
GATE	*u*	*A-operand*	*NULL*	*one of the qualifiers*
	nu	*A-operand*	*NULL*	*listed: u, nu, sf, se,*
	sf	*A-operand*	*NULL*	*snf, or sne is one of*
	se	*A-operand*	*NULL*	*a pair of statements*
	snf	*A-operand*	*NULL*	
	sne	*A-operand*	*NULL*	
	bne	A-operand	NULL	*a pair of statements*
LEAVE	leave	A-operand	B-operand	
QUEUE	queue	A-operand	B-operand	
RELEASE	release	A-operand	NULL	
SEIZE	seize	A-operand	NULL	
TRANSFER	br	NULL	B-operand	*unconditional branch*
TEST	*eq*	*A-operand*	*B-operand*	*one of the qualifiers*
	lt	*A-operand*	*B-operand*	*listed: eq, lt, or gt*
	gt	*A-operand*	*B-operand*	*is a statement along*
	bne	A-operand	NULL	*with a bne statement*
TERMINATE	terminate	A-operand	NULL	

4.3. Simulator.

One of the tools needed to implement the simulator was a random number generator. A VAX/VMS C random number utility was available, but had to meet a certain set of guidelines in order to be used in the simulator. An important factor to be considered was the period of repetition of the random number. The period was determined to be acceptable at 2^{32}. Another factor to be considered, in the GPSS application, was the number generated should be between 0.0 inclusive and 1.0 exclusive. The random number was between 0 and 2^{31}-1, but a function was created to convert the number to the appropriate range. Last but not least, the random numbers had to be evenly distributed. The C utility provided no way to pick a variable number of seeds thus one seed had to provide the stream of random numbers to be used in the simulator. A test program, contained in *Appendix E* on page 213, verified a uniform distribution choosing every fifth number for a 10,000 number sampling.

To design the simulator, the chain concept of the GPSS language was researched. Due to the subset of the language chosen, the only two chains necessary would be the Current and Future Events chains. The Future Events chain is a list of transactions whose block departure times (BDT) are greater than the absolute clock. The transactions are sorted on the chain by the block departure time. For this thesis application, the Future Events chain was used for the **ADVANCE** and **GENERATE** statements because these blocks were the only ones with associated block departure times. The Current Events chain, which is also a list of transactions, is not sorted. The Current Events chain is serviced on a first-come first-served basis. A priority scheme can also be attached to the Current Events servicing. The decision was made that none of the blocks had priority over the other therefore the current events chain is serviced with one sequential pass. An example of how these two chains interact is given below.

> The simulation begins by searching the pseudo code for all **GENERATE** statements. A transaction is generated for each statement found and the transaction is placed on the Future Events chain. Each transaction on the Future Events chain has an associated block departure time (BDT). The block departure time indicates when transaction will be scheduled to enter the model. In the next step, the simulator checks the BDT of the transaction on the top of the ordered Future Events chain and changes the simulation clock to correspond to this time. The transaction is then transferred from the Future to the Current Events chain. Another transaction is then generated for the transferred **GENERATE** statement and placed on the Future Events chain. The simulator then processes the transaction through the GPSS blocks until it can't move any further. An **ADVANCE** or **SEIZE** block are an example of a block which causes a transaction to be unable to move to the next logical block. The next consecutive transaction in the Current Events chain is processed through the model until it can't move any further. Finally, when all transactions in the Current Events chain have been processed, the simulator returns to the Future Events chain and starts the circular process of transferring a transaction onto the Current Events chain. It should be noted the only block statement which can cause a transaction to move from the Current to the Future Events chain is the **ADVANCE** statement. The **ADVANCE** statement delays the transaction by a specified amount of time, the delay is considered a BDT thus the logical association to the Future Events chain.

The C structure designed to represent the transaction is used by both the Future and Current Events chain.

```
struct  EVENTS
    {
    struct EVENTS *for_link  ;
    struct EVENTS *bck_link  ;
```

```
struct CODE   *cur_block ;
int           BDT ;
int           truth_bit ;
FAC_STATS     *ptr_fac ;
int           time_sez ;
QUE_STATS     *ptr_que ;
int           time_que ;
STOR_STATS    *ptr_stor ;
int           time_stor ;
int           blk_from ;
int           param[10] ;
};
```

The powerful data structure capability of C allows a concise description of the transaction information, and allows precise control over the memory management space within the Future and Current Events chain. The **for_link** and **bck_link** are the pointers which create the double linked list structure for the Current and Future Events chains. **cur_block** points to the current pseudo code statement, block, the transaction is executing. **BDT**, the block departure time, is only used by the Future Events chain. The **truth_bit**, use only by the Current Events chain, is used to hold the YES/NO answer from the pseudo code comparison statements EQ, LT, GT, U, NU, SF, SNF, SE, and SNE. **ptr_fac**, **ptr_que**, and **ptr_stor** are pointers to the statistics area for the model. **time_sez**, **time_que**, **time_stor** are simulation times when the transaction has entered the SEIZE, QUEUE, or ENTER block respectively. **blk_from** holds the number of the last block statement executed. Each transaction has ten arithmetic parameters associated, **param[10]**.

5. Interface Specifications.

5.1. External Interfaces.

The command line, which enters the GPSS program into the simulator, is the ***only*** authorized user interface. The format of the command is:

$ gpss *filename.extension*

No keyboard entry is allowed to interrupt the simulation once the command line is entered. If a *cntrl Y* is issued the simulation will terminate. No interactive debugging is allowed in this version of GPSS.

5.2. Internal Interfaces.

The main process of the compiler interfaces with the VMS Command Language Definition Utility. A Command Definition File was created which defines the command line, please refer to *Appendix F*, on page 216, for the command definition. Through the use of the VMS utilities, interpretation of the command line is extremely easy. Also, this facility allows easy addition of new parameters and qualifiers on the command line.

Described in *Appendix H*, on page 223, is the internal architecture of the symbol table, the pseudo code, the block statistics, the facility statistics, the queue statistics, and the storage statistics. The diagrams which describe these entities are in block form which are in one-to-one correspondence with the associated C structure. The C data structures are the key to transferring and retaining information within the internal structure of the GPSS compiler.

6. Module Design.

6.1. Processor Modules.

The following is a verbal description of the GPSS compiler/simulator. *Appendix G*, on page 217, gives a chart description of the following paragraphs.

There exists one master process, which determines whether each stage of the compiler has completed, and if the completion state allows further execution of the GPSS program.

Three divisions of the GPSS compiler/simulator exist under the master process, and each has a controlling process.

The controlling process for the parser function manages the reading of individual lines of code and determines the keyword content. Once, the keyword is determined control is given to a specific parsing routine to complete language analysis and build a skeleton pseudo code representation. These analysis routines control additions to the symbol table and the printing of messages for the syntax errors detected.

The second pass of the GPSS compiler has a controlling process which reads down the linked list of pseudo code for *holes*. *Holes* are operands whose label was not defined at the time of pseudo code generation. The correct address of the referenced label is then inserted into the operand field after location in the symbol table. The second pass of the GPSS compiler is required only for the TEST, TRANSFER, and GATE statements.

The controlling process for the simulation phase manages the initial generation of transactions and the transferring of transactions from the Future to the Current Events chain. Once, the Current events chain has been entered, each transaction steps through the pseudo code. A function in the simulator is executed for each pseudo code statement. This process is continued until the run termination count has been reached and then the statistical report is printed. The snap interval printout of the statistical report is also controlled by the main process of the simulator.

6.2. Communications among Modules

The controlling processes, described in section *6.1 Processor Modules*, page 25, are each a separate entity and when each phase has been completed passes along the symbol table address to the next phase. Program tools, because of their nature, have information passed to them through function arguments; results are then returned by the way of the RETURN() statement. In general, most other communication is done through the use of global definitions. Listed below is a list of the global definitions used through out the GPSS compiler.

Defined in the main controlling process:
- input, source program, file pointer
- line number of the source program
- block number in the GPSS program
- buffer location for the source line
- beginning pointer of the word being processed
- end pointer of the word being processed
- switch, YES/NO, for a SIMULATE statement was encountered
- switch, YES/NO, for an END statement was encountered
- total number of syntax errors
- model start information

number of start cards
number of temporary labels
first statement of the pseudo code
last statement of the pseudo code

Defined in the controlling process for pass 1:
switch, YES/NO, has a label been found on the source line
pointer to the current label's symbol table entry

Defined in the function which processes a VARIABLE statement:
the temporary variable name

Defined in the controlling process for the simulation phase:
pointer to the block statistics
pointer to the storage statistics
pointer to the queue statistics
pointer to the facility statistics
pointer to the Future Events chain
pointer to the beginning of the Current Events chain
pointer to the end of the Current Events chain
absolute clock
switch, YES/NO, snap interval print
snap interval print out time

7. Verification and Validation.

7.1. Test Plan.

Program correctness is an ambiguous term. Literature indicates testing of a program is a difficult and sometimes time consuming procedure. Proving code correctness is a bigger obstacle when the programmer has to verify his or her own program. The role of designer, programmer, and tester imbeds program functionality biases. Thus, some of the obvious tests will be overlooked due to false assumptions. The rules which were used to verify the GPSS compiler/simulator correctness were key points made in the literature read: 1. Test each field in a statement with the entire spectrum of permutations, 2. Test interrelationships of statements. Each statement separately may not produce an error but combined problems may be encountered. 3. The functions should be tested singularly, if at all possible. Bugs on a low level are easier to find.

7.1.1. Parser and Symbol Table.

All program tools used in the parser module, such as GETWORD, were tested separately before being combined with the main program interface for PASS 1.

The nineteen statements implemented in this project were tested separately and combined. A GPSS program was devised to test each each keyword statement, this included operand permutations. Refer to *Appendix B*, page 189, for the test programs. Below is a summarization of each of the nineteen test programs.

Control Statements:

SIMULATE

Test program's function:	No operands allowed for a SIMULATE statement. Two SIMULATE statements encountered. No label allowed. Extraneous information found at the end of the statement.
Permutations not tested:	Presence of control characters.

START

Test program's function:	No label allowed. Illegal A-operand. Required A-operand missing. Illegal B-operand, illegal mnemonic specification. Illegal C-operand. Extraneous information found at the end of the statement.
Permutations not tested:	Presence of control characters.

END

Test program's function:	No operands allowed for an END statement. Two END statements encountered. No label allowed. Extraneous information found at the end of the statement.
Permutations not tested:	Presence of control characters.

Entity Definition Statements:

FUNCTION

Test program's function:

Invalid label.
Incorrect specification of the random number.
Incorrect discrete function specification.
Correct function label, incorrect **FUNCTION** keyword.
X data not floating point.
Y data not integer.
Extraneous information found at the end of the statement.

Permutations not tested:

Presence of control characters.
No label specified for the **FUNCTION** statement.
Duplicate **FUNCTION** label.

STORAGE

Test program's function:

No label specified for the **STORAGE** statement.
Invalid label.
Duplicate **STORAGE** label.
Illegal A-operand.
Required A-operand missing.
Extraneous information found at the end of the statement.

Permutations not tested:

Presence of control characters.

VARIABLE

Test program's function:

No label specified for the **VARIABLE** statement.
Cannot use line defined variable in the variable statement.
Checked for correct variables within the arithmetic statements: SNA (Standard Numerical Attributes), number (0 through 9). If the variable in the expression was an SNA, the SNA was checked to make sure it was previously defined.
Checked for correct arithmetic expressions:
exp:: = [-]operand operator [-]operand | exp operator
where the operand is a valid variable and the operator is +, -, *, or /.
Checked for undefined SNA.
Extraneous information found at the end of the statement.

Permutations not tested:

Presence of control characters.
Invalid label.

Block Definition Statements:

ADVANCE

Test program's function:

Illegal A-operand.
Required A-operand missing.
Checked for correct A-operand, SNA, context.
Extraneous information found at the end of the statement.

Permutations not tested:

Presence of control characters.
Invalid label.
Checked for undefined SNA.
Checked for multiple labels.

ASSIGN

Test program's function:

Illegal A-operand.
Required A-operand missing.
Checked for correct A-operand, SNA, context.
Illegal B-operand.
Invalid B-operand arithmetic operation.
Checked for correct B-operand, SNA, context.
Checked for undefined SNA.
Extraneous information found at the end of the statement.

Permutations not tested:

Presence of control characters.
Invalid label.
Checked for multiple labels.

DEPART

Test program's function:

Illegal A-operand.
Required A-operand missing.
Checked for correct A-operand, SNA, context.
Illegal B-operand.
Checked for correct B-operand, SNA, context.
Checked for undefined SNA.
Extraneous information found at the end of the statement.

Permutations not tested:

Presence of control characters.
Invalid label.
Checked for multiple labels.

ENTER

Test program's function:

Illegal A-operand.
Required A-operand missing.
Checked for correct A-operand, SNA, context.
Illegal B-operand.
Checked for correct B-operand, SNA, context.
Checked for undefined SNA.
Checked for multiple labels.
Extraneous information found at the end of the statement.

Permutations not tested:

Presence of control characters.
Invalid label.

GATE

Test program's function:

Illegal mnemonic A-operand.
Illegal B-operand.
Checked for correct B-operand, SNA, context in comparison to the A-operand mnemonic.
Illegal C-operand.

Permutations not tested:

Presence of control characters.
Invalid label.
Checked for multiple labels.
Checked line for A-operand and NO optional B-operand.
Required A-operand missing.
Required B-operand missing.
Extraneous information found at the end of the statement.

GENERATE

Test program's function:
- Illegal A-operand.
- Required A-operand missing.
- Checked for correct A-operand, SNA, context.
- Illegal B-operand.
- Checked for correct B-operand, SNA, context.
- Checked for undefined operand.
- Extraneous information found at the end of the statement.

Permutations not tested:
- Presence of control characters.
- No label allowed.

LEAVE

Test program's function:
- Illegal A-operand.
- Required A-operand missing.
- Checked for correct A-operand, SNA, context.
- Checked for correct B-operand, SNA, context.
- Checked for multiply defined label.
- Extraneous information found at the end of the statement.

Permutations not tested:
- Presence of control characters.
- Invalid label.
- Illegal B-operand.

QUEUE

Test program's function:
- Illegal A-operand.
- Required A-operand missing.
- Checked for correct A-operand, SNA, context.
- Checked for correct B-operand, SNA, context.
- Checked for multiply defined label.
- Extraneous information found at the end of the statement.

Permutations not tested:
- Presence of control characters.
- Invalid label.
- Illegal B-operand.

RELEASE

Test program's function:
- Invalid label.
- Illegal A-operand.
- Required A-operand missing.
- Checked for correct A-operand, SNA, context.
- Extraneous information found at the end of the statement.

Permutations not tested:
- Presence of control characters.
- Checked for multiply defined label.

SEIZE

Test program's function:
- Invalid label.
- Illegal A-operand.
- Required A-operand missing.
- Checked for correct A-operand, SNA, context.
- Extraneous information found at the end of the statement.

Permutations not tested:
- Presence of control characters.
- Checked for multiply defined label.

TERMINATE

Test program's function:
- Label but no keyword.
- Illegal A-operand.
- Checked for correct A-operand, SNA, context.
- Extraneous information found at the end of the statement.

Permutations not tested:
- Presence of control characters.
- Invalid label.
- Checked for multiply defined label.

TEST

Test program's function:	Illegal A-operand mnemonic specification. Illegal C-operand. Required B-operand missing. Checked for correct B-operand, SNA, context. Illegal C-operand. Checked for correct C-operand, SNA, context. Extraneous information found at the end of the statement.
Permutations not tested:	Presence of control characters. Invalid label. Checked for multiply defined label. Checked for correct D-operand label specification.

TRANSFER

Test program's function:	Required B-operand missing. Extraneous information found at the end of the statement.
Permutations not tested:	Presence of control characters. Invalid label. Checked for multiply defined label.

A representative GPSS program was written to test the interaction between the nineteen statements. Refer to *Appendix C*, page 207, for the test program. Below is a summarization of this test program.

TRAFFIC.GPS

Test programs function:	Test interaction of nineteen GPSS statements.
Permutations not tested:	Presence of control characters. Test all options of each of the nineteen statements for interrelationship errors.

The symbol table was printed out after each test program to verify the contents. Refer to *7.3.1. Parser and Symbol Table* test results, page 39 through 81, for the symbol table displays.

7.1.2. Pseudo Code Generation.

The redesign of the parser module, due to the addition of the pseudo code skeleton generation, required the GPSS subset be tested for parser consistency. The same test programs as the parser phase (*Appendix B*, page 189, and *Appendix C*, page 207) were used for two purposes: 1. Verify the parser , 2. Test the production of pseudo code, and output the program temporary variables which were added to the symbol table. Refer to *7.3.2. Pseudo Code Generation* test results, page 82 through 158, for the pseudo code generated displays.

7.1.3. Simulator and Statistical Output.

The simulator required testing to verify the statistical report and to verify the internal functionality of the chains.

Test Case 1

Test program's function:	Test evaluation of **FUNCTION** statement. Test **GENERATE** statement with only A-operand. Test TERMINATE statement. Test termination count of 1.

Test random number generator.
Test transaction transfer from Future to Current Events chain.

Test Case 2

Test program's function: Test evaluation of **FUNCTION** statement.
Test **GENERATE** statement with only A-operand.
Test **TERMINATE** statement.
Test termination count of 20.
Test random number generator.
Test transaction transfer from Future to Current Events chain.

Test Case 3

Test program's function: Test evaluation of **FUNCTION** statement.
Test **GENERATE** statement with only A-operand.
Test **TERMINATE** statement.
Test termination count of 100.
Test random number generator.
Test transaction transfer from Future to Current Events chain.

Test Case 4

Test program's function: Test evaluation of **FUNCTION** statement.
Test **GENERATE** statement with a function A-operand and a numeric B-operand.
Test **TERMINATE** statement.
Test termination count of 1.
Test random number generator.
Test transaction transfer from Future to Current Events chain.

Test Case 5

Test program's function: Test evaluation of **FUNCTION** statement.
Test **GENERATE** statement with a function A-operand and a numeric B-operand.
Test **TERMINATE** statement.
Test termination count of 20.
Test random number generator.
Test transaction transfer from Future to Current Events chain.

Test Case 6

Test program's function: Test evaluation of **FUNCTION** statement.
Test **GENERATE** statement with a function A-operand and a numeric B-operand.
Test **TERMINATE** statement.
Test termination count of 100.
Test random number generator.
Test transaction transfer from Future to Current Events chain.

Test Case 7

Test program's function: Test evaluation of **FUNCTION** statement.
Test **GENERATE** statement with a function A-operand and a function B-operand.
Test **TERMINATE** statement.
Test termination count of 1.
Test random number generator.

Test transaction transfer from Future to Current Events chain.

Test Case 8

Test program's function: Test evaluation of FUNCTION statement.
Test GENERATE statement with a function A-operand and a function B-operand.
Test TERMINATE statement.
Test termination count of 20.
Test random number generator.
Test transaction transfer from Future to Current Events chain.

Test Case 9

Test program's function: Test evaluation of FUNCTION statement.
Test GENERATE statement with a function A-operand and a function B-operand.
Test TERMINATE statement.
Test termination count of 100.
Test random number generator.
Test transaction transfer from Future to Current Events chain.

Test Case 10

Test program's function: Test evaluation of FUNCTION statement.
Test GENERATE statement with a function A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 100.
Test random number generator.
Test a model with two GENERATE statements.
Test transaction transfer from Future to Current Events chain.

Test Case 11

Test program's function: Test evaluation of VARIABLE statement.
Test GENERATE statement with a variable A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 1.
Test transaction transfer from Future to Current Events chain.

Test Case 12

Test program's function: Test evaluation of VARIABLE statement.
Test GENERATE statement with a variable A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 20.
Test transaction transfer from Future to Current Events chain.

Test Case 13

Test program's function: Test evaluation of VARIABLE statement.
Test GENERATE statement with a variable A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 100.

Test transaction transfer from Future to Current Events chain.

Test Case 14

Test program's function:
Test GENERATE statement with a numeric A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 100.
Test a model with two GENERATE statements.
Test an ADVANCE statement with a parameter A-operand.
Test the ASSIGN statement with an addition of a numeric.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 15

Test program's function:
Test GENERATE statement with a numeric A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 100.
Test an ADVANCE statement with a parameter A-operand.
Test two ASSIGN statements where one adds a numeric and the other subtracts a numeric.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 16

Test program's function:
Test GENERATE statement with a variable A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 100.
Test an ADVANCE statement with a variable A-operand.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 17

Test program's function:
Test GENERATE statement with a numeric A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 100.
Test a model with two GENERATE statements.
Test an ASSIGN statement with an addition of a numeric.
Test an ADVANCE statement with a parameter A-operand.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 18

Test program's function:
Test GENERATE statement with a constant A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 100.

Test an ADVANCE statement with a numeric A-operand.
Test an ASSIGN statement with an addition of a numeric.
Test an ENTER statement with a variable B-operand.
Test a LEAVE statement with a numeric B-operand.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 19

Test program's function: Test GENERATE statement with a constant A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 100.
Test an ADVANCE statement with a numeric A-operand.
Test an ASSIGN statement with an addition of a numeric.
Test a QUEUE statement with a variable B-operand.
Test a DEPART statement with a numeric B-operand.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 20

Test program's function: Test GENERATE statement with a constant A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 20.
Test an ADVANCE statement with a numeric A-operand.
Test an ASSIGN statement with an addition of a numeric.
Test a SEIZE statement.
Test a RELEASE statement.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 21

Test program's function: Test GENERATE with a constant A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 20 and a snap interval of 10.
Test an ASSIGN statement with an addition of a numeric.
Test a TEST statement where the A-operand is a parameter, the B-operand a numeric, and the C-operand, a label, is specified.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 22

Test program's function: Test GENERATE with a constant A-operand and no B-operand.
Test a model with two GENERATE statements.
Test TERMINATE statement.
Test termination count of 20.
Test two ADVANCE statements with a numeric A-operand.
Test an ASSIGN statement with an addition of a numeric.

Test a SEIZE statement.
Test a RELEASE statement.
Test a GATE U statement where the A-operand is a facility, and the B-operand, a label, is specified.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 23

Test program's function: Test GENERATE with a constant A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 20.
Test two ADVANCE statements with a numeric A-operand.
Test an ASSIGN statement with an addition of a numeric.
Test a SEIZE statement.
Test a RELEASE statement.
Test a GATE NU statement where the A-operand is a facility, and the B-operand, a label, is specified.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 24

Test program's function: Test GENERATE with a constant A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 20.
Test two ADVANCE statements with a numeric A-operand.
Test an ASSIGN statement with an addition of a numeric.
Test a ENTER statement.
Test a LEAVE statement.
Test a GATE SE statement where the A-operand is a storage location, and the B-operand, a label, is specified.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 25

Test program's function: Test GENERATE with a constant A-operand and no B-operand.
Test TERMINATE statement.
Test termination count of 20.
Test two ADVANCE statements with a numeric A-operand.
Test an ASSIGN statement with an addition of a numeric.
Test a ENTER statement.
Test a LEAVE statement.
Test a GATE SF statement where the A-operand is a storage location, and the B-operand, a label, is specified.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 26

Test program's function: Test GENERATE with a constant A-operand and no

B-operand.
Test **TERMINATE** statement.
Test termination count of 20.
Test two **ADVANCE** statements with a numeric A-operand.
Test an **ASSIGN** statement with an addition of a numeric.
Test a **ENTER** statement.
Test a **LEAVE** statement.
Test a **GATE SNF** statement where the A-operand is a storage location, and the B-operand, a label, is specified.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Test Case 27

Test program's function: Test **GENERATE** with a constant A-operand and no B-operand.
Test **TERMINATE** statement.
Test termination count of 20.
Test two **ADVANCE** statements with a numeric A-operand.
Test an **ASSIGN** statement with an addition of a numeric.
Test a **ENTER** statement.
Test a **LEAVE** statement.
Test a **GATE SNE** statement where the A-operand is a storage location, and the B-operand, a label, is specified.
Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

Final Test Case

Test program's function: Test **GENERATE** with a constant A-operand and a function B-operand.
Test a model with three **GENERATE** statements.
Test **TERMINATE** statement.
Test two **ADVANCE** statements with a numeric A-operand.
Test an **ASSIGN** statement with an addition of a numeric.
Test a **ENTER** statement.
Test a **LEAVE** statement.
Test a **QUEUE** statement.
Test a **DEPART** statement.
Test a **SEIZE** statement.
Test a **RELEASE** statement.
Test a **TRANSFER** statement.
Test a **TEST G** statement where the A-operand is a parameter, the B-operand is a constant, and the C-operand, a label, is specified.

Test a **GATE U** statement where the A-operand is a facility, and the B-operand, a label, is specified.
Test a **GATE SE** statement where the A-operand is a storage location, and the B-operand, a label, is specified.Test transaction transfer from Future to Current Events chain.
Test transaction transfer from Current to Future Events chain.

All test cases were executed on the Rochester Institute of Technology GPSS

compiler running on a VAX/VMS 11/780. The results were used as a validation procedure for the thesis output.

7.2. Test Procedures.

7.2.1. Parser and Symbol Table.

Each test program, listed in *Appendix B* and *Appendix C*, was run against the first pass of the GPSS compiler/simulator. A command file was used to expedite the gathering of the verification output. By assigning SYS$OUTPUT to a file called phase1.tst, as shown in the generic example below, the results were generated:

```
$ assign/user phase1.tst sys$output
$ gpss filename.gps
```

The actual output has been included in the test results.

7.2.2. Pseudo Code Generator.

Each test program, listed in *Appendix B* and *Appendix C*, was run against the second pass of the GPSS compiler/simulator. A command file was used to expedite the gathering of the verification output. By assigning SYS$OUTPUT to a file called phase2.tst, as shown in the generic example below, the results were generated:

```
$ assign/user phase2.tst sys$output
$ gpss filename.gps
```

The actual output has been included in the test results.

7.2.3. Simulator and Statistical Output.

The statistical output contained in section *7.3.3. Simulator and Statistical Output* was obtained by executing the simulator for each test case. R.I.T.'s statistical output and the thesis statistical output was placed on the same page allowing easy comparisons of the simulations results. Also, any discrepancies in output are explained above the statistical output.

As can be seen in the test case results, a comparison of the absolute clock times for many of the programs differ. The difference in the philosophy of the two simulators is the thesis absolute clock time is allowed to circulate through the current events chain on the end termination count. R.I.T.'s, on the other hand, gets to the end termination count and immediately exits the program. Also, the R.I.T. simulator, for many of the programs, has a transaction in the GENERATE current block statistics. The GENERATE statement in this thesis works per XEROX GPDS standards which read:

> *After the BDT has been calculated, the transaction is stored on the future events chain. When its scheduled entry time (BDT) arrives, the transaction is then moved to the current events chain.*[11]

The transaction is ***not*** calculated into the current block statistics until it enters the current events chain.

7.3. Test Results.

7.3.1. Parser and Symbol Table.

Control Statements:

SIMULATE

The parser correctly tested for no operands allowed for a **SIMULATE** statement, two **SIMULATE** statements encountered, no label allowed, and extraneous information found at the end of the statement.

START

The parser correctly tested for no label allowed, illegal A-operand, required A-operand missing, illegal B-operand mnemonic specification, illegal C-operand, and extraneous information found at the end of the statement.

END

The parser correctly tested for no operands allowed for an **END** statement, two **END** statements encountered, no label allowed, and extraneous information found at the end of the statement.

Entity Definition Statements:

FUNCTION

The parser correctly tested for invalid label, incorrect specification of the random number, incorrect discrete function specification, correct function label, incorrect **FUNCTION** keyword, X data not floating point, Y data not integer, and extraneous information found at the end of the statement.

STORAGE

The parser correctly tested for no label specified for the **STORAGE** statement, invalid label, duplicate **STORAGE** label, illegal A-operand, required A-operand missing, and extraneous information found at the end of the statement.

VARIABLE

The parser correctly tested for no label specified for the **VARIABLE** statement, cannot use line defined variable in the variable statement, checked for correct variables within the arithmetic statements: SNA (Standard Numerical Attributes), number (0 through 9), if the variable in the expression was an SNA, the SNA was checked to make sure it was previously defined, checked for correct arithmetic expressions: *exp:: = [-]operand operator [-]operand | exp operator*; where the operand is a valid variable and the operator is +, -, *, or /, checked for undefined SNA, and extraneous information found at the end of the statement.

Block Definition Statements:

ADVANCE

The parser correctly tested for an illegal A-operand, required A-operand missing, checked for correct A-operand, SNA, context, and extraneous information found at the end of the statement.

ASSIGN

The parser correctly tested for an illegal A-operand, required A-operand missing,

checked for correct A-operand, SNA, context, illegal B-operand, invalid B-operand arithmetic operation, checked for correct B-operand, SNA, context, checked for undefined SNA, and extraneous information found at the end of the statement.

DEPART

The parser correctly tested for an illegal A-operand, required A-operand missing, checked for correct A-operand, SNA, context, illegal B-operand, checked for correct B-operand, SNA, context, checked for undefined SNA, and extraneous information found at the end of the statement.

ENTER

The parser correctly tested for an illegal A-operand, required A-operand missing, checked for correct A-operand, SNA, context, illegal B-operand, checked for correct B-operand, SNA, context, checked for undefined SNA, checked for multiple labels, and extraneous information found at the end of the statement.

GATE

The parser correctly tested for an illegal mnemonic A-operand, illegal B-operand, checked for correct B-operand, SNA, context in comparison to the A-operand mnemonic, and illegal C-operand.

GENERATE

The parser correctly tested for an illegal A-operand, required A-operand missing, checked for correct A-operand, SNA, context, illegal B-operand, checked for correct B-operand, SNA, context, checked for undefined operand, and extraneous information found at the end of the statement.

LEAVE

The parser correctly tested for an illegal A-operand, required A-operand missing, checked for correct A-operand, SNA, context, checked for correct B-operand, SNA, context, checked for multiply defined label, and extraneous information found at the end of the statement.

QUEUE

The parser correctly tested for an illegal A-operand, required A-operand missing, checked for correct A-operand, SNA, context, checked for correct B-operand, SNA, context, checked for multiply defined label, and extraneous information found at the end of the statement.

RELEASE

The parser correctly tested for an invalid label, illegal A-operand, required A-operand missing, checked for correct A-operand, SNA, context, and extraneous information found at the end of the statement.

SEIZE

The parser correctly tested for an invalid label, illegal A-operand, required A-operand missing, checked for correct A-operand, SNA, context, and extraneous information found at the end of the statement.

TERMINATE

The parser correctly tested for a label but no keyword, illegal A-operand, checked for correct A-operand, SNA, context, and extraneous information found at the end of the statement.

TEST

The parser correctly tested for an illegal A-operand mnemonic specification, illegal C-operand, required B-operand missing, checked for correct B-operand, SNA, context, illegal C-operand, checked for correct C-operand, SNA, context, and extraneous information found at the end of the statement.

TRANSFER

The parser correctly tested for a required B-operand missing, and extraneous information found at the end of the statement.

TRAFFIC.GPS

The parser correctly tested the nineteen GPSS statements.

*** NOTE the following pages are the parser and symbol table test results. Due to printing capabilities of the hardware, an underscore is an undefined character. The ← character is the representation for the underscore.

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for ADVANCE statement
3               !
4                   simulate
5               1       storage 1
6               var1    variable 1
7               func1   function r,d2
8               0.1,1/0.2,2
9               !
10      1       label   advance                                 !required operand missing
                                *
                >> Required operands missing
11      2               advance P1                              !illegal operand
                              *
                >> Operand is not valid
12      3               advance p1                              !illegal operand
                              *
                >> Operand is not valid
13      4               advance p*1                             !ok, parameter
14      5               advance P*1                             !ok, parameter
15      6               advance s*1                             !not allowed SNA
                                *
                >> Operand is not valid
16      7               advance S*1                             !not allowed SNA
                                *
                >> Operand is not valid
17      8               advance v*var1                          !ok, variable
18      9               advance V*var1                          !ok, variable
19      10              advance f*1                             !invalid specification of function
                                *
                >> Operand is not valid
20      11              advance fn*func1                        !ok, function
21      12              advance FN*func1                        !ok, function
22      13              advance 1                               !ok
23      14              advance a                               !illegal operand
                              *
                >> Operand is not valid
24      15              advance ,                               !required operand missing
                              *
                >> Required operands missing
25      16              advance 1,                              !syntax error
                              *
                >> Syntax error
26      17              advance 400                             !ok
27              ,       advance                                 !syntax error
                    *
                >> Syntax error
28              #$%     advance                                 !illegal label
                    *
                >> Syntax error
29      18              advance 1       1                       !syntax error
                                  *
                >> Syntax error
30                  end


**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
```

```
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label
        Label attribute: LABEL
**************************************************


Total number of errors   12
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for ASSIGN statement
3               !
4                   simulate
5               1       storage         1
6               var1    variable        1
7               func1   function        r,d2
8               0.1,1/0.2,2
9               !
10      1           assign      1,-1                        !ok
11      2           assign      p*1,+2                      !ok
12      3           assign      fn*func1,-1                 !ok
13      4           assign      v*var1,+2                   !ok
14      5           assign      1,+p*1                      !ok
15      6           assign      1,+fn*func1                 !ok
16      7           assign      1,-v*var1                   !ok
17      8           assign      ,-2                         !required operand missing
                                *
                >> Required operands missing
18      9           assign      1,+-3                       !invalid specification of arithmetic
                                  *
                >> Operand is not valid
19      10          assign      +1,2                        !invalid A-operand
                               *
                >> Operand is not valid
20      11          assign      1+,2                        !comma not in proper place
                                *
                >> Required operands missing
21      12          assign      1,                          !no numeric specified
                                    *
                >> Syntax error
22      13          assign      a,-1                        !illegal A-operand
                                *
                >> Operand is not valid
23      14          assign      1,+b                        !illegal 8-operand
                                  *
                >> Operand is not valid
24      15          assign      s*1,+1                      !illegal SNA
                                  *
                >> Operand is not valid
25      16          assign      1,-s*1                      !illegal SNA
                                    *
                >> Operand is not valid
26      17          assign      v*var2,+1                   !undefined operand
                                    *
                >> Undefined operand
                                    *
                >> Operand is not valid
27      18          assign      1,+v*var2                   !undefined operand
                                      *
                >> Undefined operand
                                      *
                >> Operand is not valid
28                  end

**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
```

```
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2
**************************************************


Total number of errors = 13
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0         May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for DEPART statement
3               !
4                   simulate
5               a       variable        1
6               b       variable        1
7               stor1   storage         20
8               func1   function        r,d2
9               0,1,1/0.2,2
10              !
11      1               depart  s*stor1                !not a good queue definition
                                  *
                >> Operand is not valid
12      2       label   depart                          !required operand missing
                          *
                >> Required operands missing
13      3               depart  1                       !ok
14      4               depart  a                       !A-operand not valid
                                  *
                >> Operand is not valid
15      5               depart  a,                      !missing B-operand
                                  *
                >> Operand is not valid
16      6               depart                          !required operand missing
                                  *
                >> Required operands missing
17      7               depart  a,b                     !illegal specification of SNA's
                                  *
                >> Operand is not valid
18      8               depart  v*a,v*b                 !ok, variable specifications
19      9               depart  p1                      !illegal operand
                                  *
                >> Operand is not valid
20              label   departr 1                       !multiple labels
                         *
                >> Multiply defined label
21      10              depart  1,v*b                   !ok
22      11              depart  v*a,2                   !ok
23      12              depart  s*1                     !not defined SNA
                                  *
                >> Undefined operand
                                  *
                >> Operand is not valid
24      13              depart  p*1,1                   !ok, correct specification
25                  end


**********************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: a
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: b
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: stor1
        Label attribute: STORAGE
        Length of information: 20

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
```

```
        Length of information: 2

Label: label
        Label attribute: LABEL
**************************************************

                .

Total number of errors = 10
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for END statement
3               !       FIRST of THREE test routines
4               !
5                   simulate
6                       end     a                       !no operands allowed
                              *
                >> Operand is not valid
7                       endd                            !misspelling take as label
                            *
                >> Label valid but no statement
8                       end                             !duplicate end
                            *
                >> Multiple END statements
9               label   end                             !no label allowed
                              *
                >> Label not allowed


************************************************

Symbol Table Contents after First Pass Completion
================================================

Label: label
        Label attribute: LABEL
************************************************


Total number of errors = 4
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *

*       GPSS V1.0         May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for END statement
3               !       SECOND of THREE test routines
4               !
5                   simulate
6                       end     ,b                      !syntax error due to comma
                              *
                >> Operand is not valid
7               #$*     end                             !illegal label
                    *
                >> Syntax error
8               .       end                             !syntax error
                     *
                >> Syntax error

Total number of errors   3
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *

*       GPSS V1.0         May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for END statement
3               !       THIRD of THREE test routines
4                   simulate
5                   end                                         !ok
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for ENTER statement
3               !
4                   simulate
5               a       variable        1
6               b       variable        1
7               stor1   storage         20
8               func1   function        r,d2
9               0.1,1/0.2,2
10              !
11      1               enter   s*stor1                  !ok, storage label specification
12      2       label   enter                           !required operand missing
                                *
                >> Required operands missing
13      3               enter   1                       !ok
14      4               enter   a                       !A-operand not valid
                                *
                >> Operand is not valid
16      5               enter   a,                      !B-operand missing
                                *
                >> Operand is not valid
16      6               enter   ,                       !required operand missing
                                *
                >> Required operands missing
17      7               enter   a,b                     !illegal specification of SNA's
                                *
                >> Operand is not valid
16      8               enter   v*a,v*b                 !ok, variable specifications
19      9               enter   p1                      !illegal operand
                                 *
                >> Operand is not valid
20              label   enterr  1                       !multiple labels
                         *
                >> Multiply defined label
21      10              enter   1,v*b                   !ok
22      11              enter   v*a,2                   !ok
23      12              enter   s*1                     !not defined SNA
                                  *
                >> Undefined operand
                                  *
                >> Operand is not valid
24      13              enter   p*1,1                   !ok, correct specification
26                  end

**********************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: a
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: b
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: stor1
        Label attribute: STORAGE
        Length of information: 20

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2
```

```
Label: label
        Label attribute: LABEL
************************************************


Total number of errors   9
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for function statement
3               !
4                   simulate
5               func1   function        r1,d2           !incorrect specification of random number
                                    *
                >> Operand is not valid
6               .5,5/.8,8                               !error on line due to faulty function definitio
**n
                    *
                >> Syntax error
7               1func   function        r1,d2           !label error
                                *
                >> Illegal specification of label
8               .5,5/.8,8                               !error on line due to faulty function definitio
**n
                    *
                >> Syntax error
9               func2   functi          r,d2            !illegal function specification
                                *
                >> Syntax error
10              .5,5/.8,8                               !error on line due to faulty function definitio
**n
                    *
                >> Syntax error
11              func3   function        r,d             !illegal specification of discrete function
                                    *
                >> Illegal specification of numeric quantity
                                    *
                >> Illegal specification of operand
12              .5,5/.8,8                               !error on line due to faulty function definiton
**
                    *
                >> Syntax error
13              func4   function        r,d*2           !illegal specification of discrete function
                                    *
                >> Illegal specification of numeric quantity
                                    *
                >> Illegal specification of operand
14              .5,5/.8,8                               !error on line due to faulty function definitio
**n
                    *
                >> Syntax error
15              func5   function        r,d2            !ok
16              .5,5/.8,8                               !ok
17              func6   function        r,d2            !ok
18              1,2/3,4                                 !illegal specification due to random number
                    *
                >> Syntax error
19              func7   function        r,d2            !ok
20              0.5,5/.6,6                              !ok
21              func8   function        r,d2            !ok
22              .5,5/.6.7                               !typing error
                        *
                >> Syntax error
23              func9   function        r,d3            !ok
24              .1,1/.2,3/.4,7                          !ok
25              func10  function        r,d3            !ok
26              .1,1/0.4,4/0.85,7                       !ok


**********************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: func1
        Label attribute: FUNCTION

Label: 1func
        Label attribute: LABEL

Label: func2
```

```
        Label attribute: LABEL

Label: func3
        Label attribute: FUNCTION

Label: func4
        Label attribute: FUNCTION

Label: func5
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.500000        5
                0.800000        B
        Length of information: 2

Label: func6
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.000000        0
                0.000000        0
        Length of information: 2

Label: func7
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.500000        5
                0.600000        6
        Length of information: 2

Label: func8
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.500000        5
                0.600000        0
        Length of information: 2

Label: func9
        Label attribute: FUNCTION
        Number of points in the function: 3
                0.100000        1
                0.200000        3
                0.400000        7
        Length of information: 3

Label: func10
        Label attribute: FUNCTION
        Number of points in the function: 3
                0.100000        1
                0.400000        4
                0.850000        7
        Length of information: 3
**************************************************

>> No END statement encountered

Total number of errors - 15
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for GATE statement
3               !
4                   simulate
5               1       storage         1
6               var1    variable        1
7               func1   function        r,d2
8               0.1,1/0.2,2
9               !
10      1       label1  gate u          s*1,label2              !invalid SNA
                                        *
                >> Operand is not valid
11      2       label2  gate u          fn*func1,label3         !ok
12      3       label3  gate u          v*var1,label4           !ok
13      4       label4  gate u          p*1,label1              !ok
14      5       label5  gate u          1,label1                !ok
15      6               gate nu         s*1,label2              !invalid SNA
                                       *
                >> Operand is not valid
16      7               gate nu         fn*func1,label2         !ok
17      8               gate nu         v*var1,label2           !ok
18      9               gate nu         p*1,label2              !ok
19      10              gate nu         1,label1                !ok
20      11              gate sf         s*1,label1              !ok
21      12              gate sf         p*1,label1              !illegal SNA
                                       *
                >> Operand is not valid
22      13              gate sf         v*var1,label1           !illegal SNA
                                        *
                >> Operand is not valid
23      14              gate sf         fn*func1,label1         !illegal SNA
                                          *
                >> Operand is not valid
24      15              gate se         s*1,label1              !ok
25      16              gate se         p*1,label1              !illegal SNA
                                       *
                >> Operand is not valid
26      17              gate se         fn*func1,label1         !illegal SNA
                                          *
                >> Operand is not valid
27      18              gate se         v*var1,label1           !illegal SNA
                                        *
                >> Operand is not valid
28      19              gate sf         1,1                     !illegal SNA
                                      *
                >> Operand is not valid
29      20              gate sf         s*1,1                   !illegal label
                                     *
                >> Illegal specification of label
30      21              gate snf        s*1,label1              !ok
31      22              gate snf        p*1,label1              !illegal SNA
                                       *
                >> Operand is not valid
32      23              gate snf        v*var1,label1           !illegal SNA
                                        *
                >> Operand is not valid
33      24              gate snf        fn*func1,label1         !illegal SNA
                                          *
                >> Operand is not valid
34      25              gate sne        s*1,label1              !ok
35      26              gate sne        p*1,label1              !illegal SNA
                                       *
                >> Operand is not valid
36      27              gate sne        fn*func1,label1         !illegal SNA
                                          *
                >> Operand is not valid
37      28              gate sne        v*var1,label1           !illegal SNA
                                        *
                >> Operand is not valid
38      29              gate snf        1,1                     !illegal SNA
                                      *
                >> Operand is not valid
39      30              gate snf        s*1,1                   !illegal label
```

```
                >> Illegal specification of label
40      31              gate ru         1,label1                !illegal mneumonic
                >> Illegal option specified
41                  end

**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label1
        Label attribute: LABEL

Label: label2
        Label attribute: LABEL

Label: label3
        Label attribute: LABEL

Label: label4
        Label attribute: LABEL

Label: label5
        Label attribute: LABEL
**************************************************


Total number of errors : 19
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for GENERATE statement
3               !
4                   simulate
5               1       storage 1
6               var1    variable        1
7               func1   function        r,d2
8               0.1,1/0.2,2
9               !
10      1           generate    1               !ok
11      2           generate    5,1             !ok
12      3           generate    1,p*1           !can't cause no parameters are evaluated yet
                                    *
                >> Operand is not valid
13      4           generate    1,fn*func1      !ok
14      5           generate    1,v*var1        !ok
15      6           generate    p*1,1           !can't cause no parameters are evaluated yet
1                                   *
                >> Operand is not valid
6       7           generate    v*var1,1        !ok
17      8           generate    fn*func1,1      !ok
18      9           generate    s*1,1           !illegal SNA
                                    *
                >> Operand is not valid
19      10          generate    1,s*1           !illegal SNA
                                      *
                >> Operand is not valid
20      11          generate    ,1              !required operand missing
                                   *
                >> Required operands missing
21      12          generate    1,              !syntax error
                                     *
                >> Syntax error
22      13          generate    fn*func2        !undefined function
                                        *
                >> Undefined operand
                                        *
                >> Operand is not valid
23      14          generate    1       1       !syntax error
                                      *
                >> Syntax error
24      15          generate    1,1,1           !syntax error
                                        *
                >> Syntax error
2L                  end

************************************************

Symbol Table Contents after First Pass Completion
================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2
************************************************


Total number of errors - 10
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for LEAVE statement
3               !
4                   simulate
5               a       variable        1
6               b       variable        1
7               stor1   storage         20
8               func1   function        r,d2
9               0.1,1/0.2,2
10              !
11      1               leave   s*stor1                 !ok, storage label specification
12      2       label   leave                           !required operand missing
                                *
                >> Required operands missing
13      3               leave   1                       !ok
14      4               leave   a                       !illegal A-operand
                                *
                >> Operand is not valid
15      5               leave   a,                      !no B-operand
                                *
                >> Operand is not valid
16      6               leave   ,                       !required operand missing
                                *
                >> Required operands missing
17      7               leave   a,b                     !illegal specification of SNA's
                                *
                >> Operand is not valid
18      8               leave   v*a,v*b                 !ok, variable specifications
19      9               leave   p1                      !illegal operand
                                *
                >> Operand is not valid
20              label   leaver  1                       !multiple labels
                         *
                >> Multiply defined label
21      10              leave   1,v*b                   !ok
22      11              leave   v*a,2                   !ok
23      12              leave   s*1                     !not defined SNA
                                *
                >> Undefined operand
                                *
                >> Operand is not valid
24      13              leave   p*1,1                   'ok, correct specification
26                  end


**********************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: a
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: b
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: stor1
        Label attribute: STORAGE
        Length of information: 20

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2
```

```
Label: label
        Label attribute: LABEL
**************************************************


Total number of errors = 0
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for QUEUE statement
3               !
4                   simulate
5               a       variable        1
6               b       variable        1
7               stor1   storage         20
8               func1   function        r,d2
9               0.1,1/0.2,2
10              !
11      1               queue   s*stor1                 !ok, storage label specification
                                    *
                >> Operand is not valid
12      2       label   queue                           !required operand missing
                              *
                >> Required operands missing
13      3               queue   1                       !ok
14      4               queue   a                       !illegal A-operand
                                *
                >> Operand is not valid
15      5               queue   a,                      !no B-operand
                                *
                >> Operand is not valid
16      6               queue   ,                       !required operand missing
                                *
                >> Required operands missing
17      7               queue   a,b                     !illegal specification of SNA's
                                *
                >> Operand is not valid
18      8               queue   v*a,v*b                 !ok, variable specifications
19      9               queue   p1                      !illegal operand
                                 *
                >> Operand is not valid
20              label   queuer  1                       !multiple labels
                         *
                >> Multiply defined label
21      10              queue   1,v*b                   !ok
22      11              queue   v*a,2                   !ok
23      12              queue   s*1                     !not defined SNA
                                  *
                >> Undefined operand
                                *
                >> Operand is not valid
24      13              queue   p*1,1                   !ok, correct specification
25                  end


**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: a
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: b
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: stor1
        Label attribute: STORAGE
        Length of information: 20

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
```

```
        Length of information: 2

Label: label
        Label attribute: LABEL
**************************************************


Total number of errors = 10
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for RELEASE statement
3               !
4                   simulate
5               1       storage 1
6               jacky   variable        1
7               func1   function r,d2
8               0.1,1/0,2,2
9               !
10      1       label   release                         !required operand missing
                                *
                >> Required operands missing
11      2       1a      release 2                       !illegal label
                                *
                >> Illegal specification of label
12      3               release 1                       !ok
13      4               release a                       !not allowed SNA
                                *
                >> Operand is not valid
14      5               release p1                      !illegal specification of SNA
                                *
                >> Operand is not valid
15      6               release 1,                      !syntax error
                                *
                >> Syntax error
16      7               release ,1                      !required operand missing
                                *
                >> Required operands missing
17      8               release P1                      !illegal specification of SNA
                                *
                >> Operand is not valid
18      9               release p*1                     !ok, parameter
19      10              release P*1                     !ok, parameter
20      11              release s*1                     !not allowable SNA
                                *
                >> Operand is not valid
21      12              release S*1                     !not allowable SNA
                                *
                >> Operand is not valid
22      13              release v*jacky                 !ok, variable
23      14              release V*jacky                 !ok, variable
24      15              release f*1                     !facility not allowed
                                *
                >> Operand is not valid
25      16              release FN*func1                !ok, function
26      17              release fn*func1                !ok, function
27      18              release p*1                     !ok, parameter
28      19              release P*1                     !ok, parameter
29      20              release 1       1               !syntax error
                                *
                >> Syntax error
30              ,       release                         !syntax error
                  *
                >> Syntax error
31              @##     release                         !illegal label
                    *
                >> Syntax error
32                  end


************************************************

Symbol Table Contents after First Pass Completion
================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: jacky
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
```

```
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label
        Label attribute: LABEL

Label: 1a
        Label attribute: LABEL
**********************************************


Total number of errors   13
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for SEIZE statement
3               !
4                   simulate
5               1       storage 1
6               jacky   variable        1
7               func1   function r,d2
8               0.1,1/0.2,2
9               !
10      1       label   seize                           !required operand missing
                                *
                >> Required operands missing
11      2       1a      seize   2                       !illegal label
                          *
                >> Illegal specification of label
12      3               seize   1                       !ok
13      4               seize   a                       !not allowed SNA
                          *
                >> Operand is not valid
14      5               seize   p1                      !illegal specification of SNA
                          *
                >> Operand is not valid
15      6               seize   1,                      !syntax error
                          *
                >> Syntax error
16      7               seize   ,1                      !required operand missing
                          *
                >> Required operands missing
17      8               seize   P1                      !illegal specification of SNA
                          *
                >> Operand is not valid
18      9               seize   p*1                     !ok, parameter
19      10              seize   ?*1                     !ok, parameter
20      11              seize   s*1                     !not allowable SNA
                            *
                >> Operand is not valid
21      12              seize   S*1                     !not allowable SNA
                            *
                >> Operand is not valid
22      13              seize   v*jacky                 !ok, variable
23      14              seize   V*jacky                 !ok, variable
24      15              seize   f*1                     !facility not allowed
                            *
                >> Operand is not valid
25      16              seize   FN*func1                !ok, function
26      17              seize   fn*func1                !ok, function
27      18              seize   p*1                     !ok, parameter
28      19              seize   P*1                     !ok, parameter
29      20              seize   1       1               !syntax error
                            *
                >> Syntax error
30              .       seize                           !syntax error
                    *
                >> Syntax error
31              @##     seize                           !illegal label
                  *
                >> Syntax error
32                  end

**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: jacky
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
```

```
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label
        Label attribute: LABEL

Label: 1a
        Label attribute: LABEL
**********************************************


Total number of errors   13
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for SIMULATE statement
3               !       FIRST of THREE test routines
4               !
5                       simulate        a                       !no operands allowed
                                        *
                >> Operand is not valid
6                       simulatte                               !misspelling take as label
                                      *
                >> Label valid but no statement
7                       simulate                                !duplicate simulate
                                      *
                >> Multiple SIMULATION statements
8               label   simulate                                !no label allowed
                                      *
                >> Label not allowed
9                   end


**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: label
        Label attribute: LABEL
**************************************************


Total number of errors = 4
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *

*       GPSS V1.0         May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for SIMULATE statement
3               !       SECOND of THREE test routines
4               !
5                       simulate        ,b                      !syntax error due to comma
                                        *
                >> Syntax error
6               #$*     simulate                                !illegal label
                      *
                >> Syntax error
7               .       simulate                                !syntax error
                        *
                >> Syntax error
8                   end

Total number of errors - 3
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *

*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for SIMULATE statement
3               !       THIRD of THREE test routines
4               !
5                   simulate                                !ok
6                   end
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for START statement
3               !
4                   simulate
5               !
6               label   start   1                               !label not allowed
                        *
                >> Label not allowed
7               .       start                                   !syntax error
                  *
                >> Syntax error
8                       start   1                               !ok
9                       start   a                               !illegal operand
                        *
                >> Undefined operand
10                      start   ,NP,3                           !required operand missing
                        *
                >> Required operands missing
11                      start   1,np                            !ok
12                      start   1,,                             !syntax correct, just no arguements
13                      start   1,,c                            !illegal operand
                          *
                >> Undefined operand
14                      start   1,,2                            !ok
15                      start   1,NULL                          !ok
16                      start   1,bb                            !illegal option
                          *
                >> Operand is not valid
17                      start   1       np      1               !syntax error
                          *
                >> Syntax error
18                  end

************************************************

Symbol Table Contents after First Pass Completion
================================================

Label: label
        Label attribute: LABEL
************************************************


Total number of errors   7
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for STORAGE statement
3               !
4                   simulate
5               !
6                       storage                         !missing label
                                *
                >> Label required
7               a       storage 2                       !ok
8               1       storage 3                       !ok
9               1a      storage 4                       !ok
                                *
                >> Illegal specification of numeric quantity
                                *
                >> Illegal specification of label
10              1       storage 5                       !duplicate label
                          *
                >> Multiply defined label
11              2       storage                         !required operand missing
                                *
                >> Required operands missing
12              3       storage six                     !illegal operand
                                *
                >> Operand is not valid
13              4       storage 7,                      !syntax error
                                *
                >> Syntax error
14              5       storage 8,9                     !syntax error
                                *
                >> Syntax error
15              6       storage 10      11              !syntax error
                                  *
                >> Syntax error
16              .       storage                         !syntax error
                          *
                >> Syntax error
17              $%↑↑    storage 1                       !illegal label
                          *
                >> Syntax error
18                  end

************************************************

Symbol Table Contents after First Pass Completion
================================================

Label: a
        Label attribute: STORAGE
        Length of information: 2

Label: 1
        Label attribute: STORAGE
        Length of information: 3

Label: 1a
        Label attribute: STORAGE

Label: 2
        Label attribute: STORAGE

Label: 3
        Label attribute: STORAGE

Label: 4
        Label attribute: STORAGE
        Length of information: 7

Label: 5
        Label attribute: STORAGE
        Length of information: 8

Label: 6
```

```
        Label attribute: STORAGE
        Length of information: 10
**************************************************

Total number of errors = 11
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0         May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for TERMINATE statement
3               !
4                   simulate
5               1       storage 1
6               var1    variable        1
7               func1   function        r,d2
8               0.1,1/0.2,2
9               !
10      1       label   terminate                               !ok
11      2               terminate                               !ok, default 1
12      3               terminate       1                       !ok
13      4               terminate       ,                       !syntax error
                            *
                >> Syntax error
14      5               terminate       a                       !illegal operand
                            *
                >> Operand is not valid
15      6               terminate       2,                      !syntax error
                            *
                >> Syntax error
16      7               terminate       3       4               !syntax error
                            *
                >> Syntax error
17      8               terminate       5                       !ok
18                      terminatte                              !misspelling, error
                            *
                >> Label valid but no statement
19              ,       terminate                               !syntax error
                    *
                >> Syntax error
20              @##     terminate                               !illegal label
                    *
                >> Syntax error
21      9               terminate       P1                      !illegal operand
                            *
                >> Operand is not valid
22      10              terminate       p1                      !illegal operand
                            *
                >> Operand is not valid
23      11              terminate       p*1                     !no SNA's allowed
                            *
                >> Operand is not valid
24      12              terminate       P*1                     !no SNA's allowed
                            *
                >> Operand is not valid
25      13              terminate       s*1                     !no SNA's allowed
                            *
                >> Operand is not valid
26      14              terminate       S*1                     !no SNA's allowed
                            *
                >> Operand is not valid
27      15              terminate       v*1                     !no SNA's allowed
                            *
                >> Operand is not valid
28      16              terminate       V*1                     !no SNA's allowed
                            *
                >> Operand is not valid
29      17              terminate       fn*1                    !no SNA's allowed
                            *
                >> Operand is not valid
30      18              terminate       FN*1                    !no SNA's allowed
                            *
                >> Operand is not valid
31                  end

**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: 1
```

```
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label
        Label attribute: LABEL
************************************************


Total number of errors - 17
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for TEST statement
3               !
4                       simulate
5               1       storage         1
6               var1    variable        1
7               func1   function        r,d2
8               0.1,1/0.2,2
9               !
10      1       label1  test e  p*1,p*2,label2          !ok
11      2       label2  test e  v*var1,p*1,label3       !ok
12      3       label3  test e  fn*func1,p*1,label1     !ok
13      4               test e  1,p*1,label2            !ok
14      5               test e  1,2,label3              !ok
15      6               test e  1,p*1,label1            !ok
16      7               test e  1,fn*func1,label2       !ok
17      8               test e  1,v*var1,label3         !ok
18      9               test l  1,2,label3              !ok
19      10              test g  1,2,label3              !ok
20      11              test ge 1,2,label3              !illegal option
                                *
                >> Illegal option specified
21      12              test l  s*1,1,label3            !illegal SNA
                                  *
                >> Operand is not valid
22      13              test g  1,s*1,label3            !illegal SNA
                                    *
                >> Operand is not valid
23      14              test e  ,1,label3               !required operand missing
                                *
                >> Required operands missing
24      15              test l  1,,label3               !required operand missing
                                  *
                >> Required operands missing
25      16              test g  1,1                     !ok
26      17              test e  1,1,label4              !incorrect label error second pass
27      18              test l  a,1,label1              !illegal operand
                                *
                >> Operand is not valid
28      19              test g  1,a,label1              !illegal operand
                                  *
                >> Operand is not valid
29      20              test e  v*var2,p*2,label2       !undefined operand
                                    *
                >> Undefined operand
                                  *
                >> Operand is not valid
30                  end


************************************************

Symbol Table Contents after First Pass Completion
================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label1
```

```
        Label attribute: LABEL

Label: label2
        Label attribute: LABEL

Label: label3
        Label attribute: LABEL
************************************************


Total number of errors = 9
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       Purpose:
3               !               The following program is used for a generic debugging tool
4               !       for the GPSS compiler/simulator thesis.  The following program
5               !       depicts a traffic problem over a bridge where one lane has been shut
6               !       off and the traffic is redirected through the other lane by means of
7               !       a traffic light at each end of the bridge.
8               !
9               !       Written:
10              !               Jacky Van Dellon        March 15, 1984
11              !
12              !-------------------------------------------------------------------------------
13              !
14              !       Parameters:
15              !               4       - keep count of cars in direction 2
16              !               5         keep count of cars in direction 1
17              !
18                  simulate
19              !
20              !       Traffic lights problem
21              !       ======================
22              expon   function        r,d23
23              0.0,0/0.1,104/0.2,222/0.3,355/0.4,509/0.5,690/0.6,915/0.7,1200/0.8,1600/
24              0.84,1830/0.88,2120/0.9,2300/0.92,2520/0.94,2810/0.95,2990/0.96,3200/0.97,3500/
25              0.98,3900/0.99,4600/0.995,5300/0.998,6200/0.999,7000/0.9997,8000
27              gre1    variable        600     ! length of time for green light in direction 1
28              gre2    variable        450     ! length of time for green light in direction 2
29              stop    storage         1       ! storage to restrict flow of transactions
30              1       storage         30      ! storage used for entering cars in direction 1
31              2       storage         30      ! storage used for entering cars in direction 2
32              !
33              !       traffic from direction 1
34              !       ========================
35      1               generate 90,fn*expon            !generate cars
36      2               queue   1                       !enter queue
37      3               seize   1                       !seize starting place
38      4               gate u  1                       !is light green
39      5               transfer        ,delay          !null transfer
40              !
41      6       null1   advance 100                     !null block
42              !
43      7       delay   advance 20                      !start car
44      8               assign  5,+1                    !keep count of cars
45      9               enter   1,1                     !enter storage
46      10              depart  1                       !leave queue
47      11              release 1                       !release starting place
48      12              leave   1                       !leave storage
49      13              terminate                       !car is over bridge
50              !
51              !       traffic from direction 2
52              !       ========================
53      14              generate 120,fn*expon           !generate cars
54      15              queue   2                       !enter queue
55      16              seize   2                       !seize starting position
56      17              gate u  2                       !is light 2 green
57      18              test g  p*5,10,around           !check for ten cars in the counter
58              !
59      19      null2   advance 100                     !null block
60              !
61      20      around  advance 20                      !start car
62      21              assign  4,+1                    !count cars in direction 2
63      22              depart  2                       !leave starting place
64      23              release 2                       !release starting place
65      24              terminate                       !car is over bridge
66              !
67              !       traffic lights
68              !       ==============
69      25              generate 1                      !just start this sequence up
70      26              gate se s*stop                  !make sure only one goes through
71      27              enter   s*stop                  !one is through and starts cycle
72              !
73      28      loop    advance 550                     !both lights red
74      29              seize   1                       !light 1 becomes green
```

```
75      30              advance v*gre1                  !green time for 1
76      31              release 1                       !light one turns red
77      32              advance 550                     !both lights red
78      33              seize   2                       !light 2 becomes green
79      34              advance v*gre2                  !green time for 2
80      35              release 2                       !light two turns red
81      36              transfer ,loop                  !begin new light cycle
82              !
83              !       auxiliary clock
84              !       ===============
85      37              generate 36000                  !one transaction per hour
86      38              terminate 1                     !decrement
87              !
88              !       start
89              !       =====
90                  start               1
91              !
92                  end


**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: expon
        Label attribute: FUNCTION
        Number of points in the function: 23
                0.000000        0
                0.100000        104
                0.200000        222
                0.300000        355
                0.400000        509
                0.500000        690
                0.600000        915
                0.700000        1200
                0.800000        1600
                0.840000        1830
                0.880000        2120
                0.900000        2300
                0.920000        2520
                0.940000        2810
                0.950000        2990
                0.960000        3200
                0.970000        3500
                0.980000        3900
                0.990000        4600
                0.995000        5300
                0.998000        6200
                0.999000        7000
                0.999700        8000
        Length of information: 23

Label: gre1
        Label attribute: VARIABLE
        Length of information: 4
        Information stored:
                600:

Label: gre2
        Label attribute: VARIABLE
        Length of information: 4
        Information stored:
                450:

Label: stop
        Label attribute: STORAGE
        Length of information: 1

Label: 1
        Label attribute: STORAGE
        Length of information: 30

Label: 2
        Label attribute: STORAGE
        Length of information: 30
```

```
Label: null1
        Label attribute: LABEL

Label: delay
        Label attribute: LABEL

Label: null2
        Label attribute: LABEL

Label: around
        Label attribute: LABEL

Label: loop
        Label attribute: LABEL
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for an unconditional transfer
3               !
4                   simulate
5               1       storage         1
6               var1    variable        1
7               func1   function        r,d2
8               0.1,1/0.2,2
9               !
10      1               transfer        ,                       ! no label error
                                        *
                >> Required operands missing
11              ,       transfer                                ! syntax error
                    *
                >> Syntax error
12      2       label1  transfer        ,label2                 ! ok
13      3               transfer        ,label2                 ! ok
14      4               transfer        ,,                      ! syntax error
                                      *
                >> Syntax error
15                  end


************************************************

Symbol Table Contents after First Pass Completion
================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label1
        Label attribute: LA8EL

Total number of errors = 3
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for VARIABLE statement
3               !
4                   simulate
5               func1   function        r,d2
6               0.1,1/0.2,2
7               !
8               var1    variable        1*2+3-4/5               !ok
9               var2    variable        1+v*var1+3              !ok
10              var3    variable        v*var2+p*1+fn*func1     !ok
11              var4    variable        p*1-v*var3              !ok
12              var5    variable        v*var5*v*var2           !cannot use same variable
                                                *
                >> Undefined operand
13                      variable        1+2                     !required label missing
                                    *
                >> Label required
14              vara    variable        1+*2                    !arithmetic definition error
                                  *
                >> Syntax error
15              varb    variable        1+(2*3),                !arithmetic definition error
                                  *
                >> Syntax error
                                  *
                >> Operand is not valid
16              varc    variable        +1*2                    !arithmetic definition error
                                  *
                >> Illegal variable definition
17              vard    variable        22+30                   !ok
18              vare    variable        2a+3                    !illegal operand
                                  *
                >> Syntax error
19              var6    variabl         3                       !syntax error
                                    *
                >> Syntax error
20              var7    variable        a+b                     !illegal operand
                                  *
                >> Syntax error
                                  *
                >> Required operands missing
21              var8    variable        s*2+1                   !illegal SNA
                                  *
                >> Syntax error
                                  *
                >> Required operands missing
22              var9    variable        1,2                     !arithmetic definition error
                                    *
                >> Syntax error
23              var10   variable        fn*func1+2              !undefined operand
24              var11   variable        1 1                     !syntax error
                                  *
                >> Operand is not valid
25              var12   variable        1-2                     !ok
26              var13   variable        1/2                     !ok
27              var14   variable        -1                      !ok
28              var15   variable        -fn*func1               !ok
29              var16   variable        -1/2*3                  !ok
30              var17   variable        2/3+4/5+5*6*3           !ok
31              var18   variable        2/3+-4                  !ok, (2/3) + -(4)
32              var19   variable        2/v*var6+-v*var7        !ok
33              var20   variable        1--v*var13              !ok, 1- ( -(var13))
34              var21   variable        v*var13*var14           !illegal specification of variable
                                                  *
                >> Undefined operand
35              var22   variable        1/-2                    !ok
36              var23   variable        1/*23                   !arithmetic definition error
                                  *
                >> Illegal variable definition
37              var24   variable        34*-/34                 !arithmetic definiton error
                                    *
                >> Illegal variable definition
38              var25   variable        8*v*var5/v*var22        !ok
39              var26   variable        -v*var14                !ok
```

```
40              var27   variable        var14 + var15
                                        *
                >> Syntax error
                                        *
                >> Required operands missing
41              var27a  variable        zebra + wolf
                                        *
                >> Syntax error
                                        *
                >> Required operands missing
42              var28   variable        1 / 2
43              var29   variable        1+2+3+4+5+6+
                                                  *
                >> Operand is not valid
44              var30   variable        65*59/45/64
45              end


************************************************

Symbol Table Contents after First Pass Completion
================================================

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: var1
        Label attribute: VARIABLE
        Length of information: 20
        Information stored:
                v1t000:3:+:v1t001:-:

Label: v1t000
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                1:2:*:

Label: v1t001
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                4:5:/:

Label: var2
        Label attribute: VARIABLE
        Length of information: 13
        Information stored:
                1:var1:+:3:+:

Label: var3
        Label attribute: VARIABLE
        Length of information: 17
        Information stored:
                var2:1:+:func1:+:

Label: v1t002
        Label attribute: PARAMETER
        Information stored:
                1

Label: var4
        Label attribute: VARIABLE
        Length of information: 9
        Information stored:
                1:var3:-:

Label: v1t003
        Label attribute: PARAMETER
        Information stored:
                1
```

```
Label: var5
        Label attribute: VARIABLE

Label: vara
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: varb
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: varc
        Label attribute: VARIABLE

Label: vard
        Label attribute: VARIABLE
        Length of information: 8
        Information stored:
                22:30:+:

Label: vare
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                2:

Label: var6
        Label attribute: LABEL

Label: var7
        Label attribute: VARIABLE

Label: var8
        Label attribute: VARIABLE

Label: var9
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: var10
        Label attribute: VARIABLE
        Length of information: 10
        Information stored:
                func1:2:+:

Label: var11
        Label attribute: VARIABLE

Label: var12
        Label attribute: VARIABLE
        Length of information: 6
        Information stored:
                1:2:-:

Label: var13
        Label attribute: VARIABLE
        Length of information: 7
        Information stored:
                v1t004:

Label: v1t004
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                1:2:/:

Label: var14
        Label attribute: VARIABLE
        Length of information: 4
        Information stored:
```

```
                1:~:

Label: var15
        Label attribute: VARIABLE
        Length of information: 8
        Information stored:
                func1:~:

Label: var16
        Label attribute: VARIABLE
        Length of information: 7
        Information stored:
                v1t006:

Label: v1t005
        Label attribute: TEMPORARY
        Length of information: 8
        Information stored:
                1:~:2:/:

Label: v1t006
        Label attribute: TEMPORARY
        Length of information: 11
        Information stored:
                v1t005:3:*:

Label: var17
        Label attribute: VARIABLE
        Length of information: 25
        Information stored:
                v1t007:v1t008:+:v1t010:+:

Label: v1t007
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                2:3:/:

Label: v1t008
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                4:5:/:

Label: v1t009
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                5:6:*:

Label: v1t010
        Label attribute: TEMPORARY
        Length of information: 11
        Information stored:
                v1t009:3:*:

Label: var18
        Label attribute: VARIABLE
        Length of information: 13
        Information stored:
                v1t011:4:~:+:

Label: v1t011
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                2:3:/:

Label: var19
        Label attribute: VARIABLE
        Length of information: 16
        Information stored:
                v1t012:var7:~:+:

Label: v1t012
        Label attribute: TEMPORARY
```

```
        Length of information: 11
        Information stored:
                2:v*var6:/:

Label: var20
        Label attribute: VARIABLE
        Length of information: 12
        Information stored:
                1:var13:~:-:

Label: var21
        Label attribute: VARIABLE

Label: var22
        Label attribute: VARIABLE
        Length of information: 7
        Information stored:
                v1t013:

Label: v1t013
        Label attribute: TEMPORARY
        Length of information: 8
        Information stored:
                1:2:~:/:

Label: var23
        Label attribute: VARIABLE

Label: v1t014
        Label attribute: TEMPORARY
        Length of information: 2
        Information stored:
                1:

Label: var24
        Label attribute: VARIABLE

Label: v1t015
        Label attribute: TEMPORARY
        Length of information: 3
        Information stored:
                34:

Label: var25
        Label attribute: VARIABLE
        Length of information: 7
        Information stored:
                v1t017:

Label: v1t016
        Label attribute: TEMPORARY
        Length of information: 11
        Information stored:
                8:v*var5:*:

Label: v1t017
        Label attribute: TEMPORARY
        Length of information: 17
        Information stored:
                v1t016:v*var22:/:

Label: var26
        Label attribute: VARIABLE
        Length of information: 8
        Information stored:
                var14:~:

Label: var27
        Label attribute: VARIABLE

Label: var27a
        Label attribute: VARIABLE

Label: var28
        Label attribute: VARIABLE
        Length of information: 7
```

```
        Information stored:
                v1t018:

Label: v1t018
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                1:2:/:

Label: var29
        Label attribute: VARIABLE
        Length of information: 22
        Information stored:
                1:2:+:3:+:4:+:5:+:6:+:

Label: var30
        Label attribute: VARIABLE
        Length of information: 7
        Information stored:
                v1t021:

Label: v1t019
        Label attribute: TEMPORARY
        Length of information: 8
        Information stored:
                65:59:*:

Label: v1t020
        Label attribute: TEMPORARY
        Length of information: 12
        Information stored:
                v1t019:45:/:

Label: v1t021
        Label attribute: TEMPORARY
        Length of information: 12
        Information stored:
                v1t020:64:/:
**************************************************

======================
START card information
======================

Total number of errors = 22
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

8.3.2. Pseudo Code Generator.

Control Statements:

SIMULATE	No pseudo code was generated, which is correct.
START	No pseudo code was generated, which is correct.
END	No pseudo code was generated, which is correct.

Entity Definition Statements:

FUNCTION	No pseudo code was generated, which is correct.
STORAGE	No pseudo code was generated, which is correct.
VARIABLE	No pseudo code was generated, which is correct.

Block Definition Statements:

ADVANCE	Correct pseudo code was generated for the test case.
ASSIGN	Correct pseudo code was generated for the test case.
DEPART	Correct pseudo code was generated for the test case.
ENTER	Correct pseudo code was generated for the test case.
GATE	Correct pseudo code was generated for the test case.
GENERATE	Correct pseudo code was generated for the test case.
LEAVE	Correct pseudo code was generated for the test case.
QUEUE	Correct pseudo code was generated for the test case.
RELEASE	Correct pseudo code was generated for the test case.
SEIZE	Correct pseudo code was generated for the test case.
TERMINATE	Correct pseudo code was generated for the test case.
TEST	Correct pseudo code was generated for the test case.
TRANSFER	Correct pseudo code was generated for the test case.
TRAFFIC.GPS	Correct pseudo code was generated for the test case.

*** NOTE the following pages are the pseudo code generator test results. Due to printing capabilities of the hardware, an underscore is an undefined character. The ← character is the representation for the underscore.

```
*       GPSS V1.0         May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for ADVANCE statement
3               !
4                   simulate
5               1       storage 1
6               var1    variable 1
7               func1   function r,d2
8               0.1,1/0.2,2
9               !
10      1       label   advance                                 !required operand missing
                                 *
                >> Required operands missing
11      2               advance P1                              !illegal operand
                                *
                >> Operand is not valid
12      3               advance p1                              !illegal operand
                               *
                >> Operand is not valid
13      4               advance p*1                             !ok, parameter
14      5               advance P*1                             !ok, parameter
15      6               advance s*1                             !not allowed SNA
                                *
                >> Operand is not valid
16      7               advance S*1                             !not allowed SNA
                                *
                >> Operand is not valid
17      8               advance v*var1                          !ok, variable
18      9               advance V*var1                          !ok, variable
19      10              advance f*1                             !invalid specification of function
                                *
                >> Operand is not valid
20      11              advance fn*func1                        !ok, function
21      12              advance FN*func1                        !ok, function
22      13              advance 1                               !ok
23      14              advance a                               !illegal operand
                                *
                >> Operand is not valid
24      15              advance ,                               !required operand missing
                                *
                >> Required operands missing
25      16              advance 1,                              !syntax error
                               *
                >> Syntax error
26      17              advance 400                             !ok
27              ,       advance                                 !syntax error
                    *
                >> Syntax error
28              #$%     advance                                 !illegal label
                    *
                >> Syntax error
29      18              advance 1       1                       !syntax error
                                    *
                >> Syntax error
30                  end
```

```
**********************************************

Symbol Table Contents after First Pass Completion
==============================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label
        Label attribute: LABEL

Label: v1t000
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t001
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t002
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t003
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t004
        Label attribute: CONSTANT
        Information stored:
                400

Label: v1t005
        Label attribute: CONSTANT
        Information stored:
                1
**********************************************
```

```
======================
START card information
======================


====================
Pseudo Code Generated
====================


operator: ADVANCE
block number - 3
A←operand= v1t000
B←operand= * NULL *

operator: ADVANCE
block number - 4
A←operand= v1t001
B←operand= * NULL *

operator: ADVANCE
block number - 7
A←operand= var1
B←operand= * NULL *

operator: ADVANCE
block number - 8
A←operand= var1
B←operand= * NULL *

operator: ADVANCE
block number - 10
A←operand= func1
B←operand= * NULL *

operator: ADVANCE
block number = 11
A←operand= func1
B←operand= * NULL *

operator: ADVANCE
block number = 12
A←operand= v1t002
B←operand= * NULL *

operator: ADVANCE
block number   16
A←operand= v1t004
B←operand= * NULL *

Total number of errors   12
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS exiting    *
*       GPSS V1.0         May,1984
*     author: J. Van Dellon
1                !
2                !       parsing test for ASSIGN statement
3                !
4                    simulate
5                1       storage         1
6                var1    variable        1
7                func1   function        r,d2
8                0.1,1/0.2,2
9                !
10       1           assign      1,-1                        !ok
11       2           assign      p*1,+2                      !ok
12       3           assign      fn*func1,-1                 !ok
13       4           assign      v*var1,+2                   !ok
14       5           assign      1,+p*1                      !ok
15       6           assign      1,+fn*func1                 !ok
16       7           assign      1,-v*var1                   !ok
17       8           assign      ,-2                         !required operand missing
                                 *
                 >> Required operands missing
18       9           assign      1,+-3                       !invalid specification of arithmetic
                                   *
                 >> Operand is not valid
19       10          assign      +1,2                        !invalid A-operand
                                *
                 >> Required operands missing
20       11          assign      1+,2                        !comma not in proper place
                                 *
                 >> Required operands missing
21       12          assign      1,-                         !no numeric specified
                                    *
                 >> Syntax error
22       13          assign      a,-1                        !illegal A-operand
                                 *
                 >> Operand is not valid
23       14          assign      1,+b                        !illegal B-operand
                                   *
                 >> Operand is not valid
24       15          assign      s*1,+1                      !illegal SNA
                                   *
                 >> Operand is not valid
25       16          assign      1,-s*1                      !illegal SNA
                                    *
                 >> Operand is not valid
26       17          assign      v*var2,+1                   !undefined operand
                                    *
                 >> Undefined operand
27       18          assign      1,+v*var2                   !undefined operand
                                     *
                 >> Undefined operand
                                     *
                 >> Operand is not valid
28                   end
```

```
**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: v1t000
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t001
        Label attribute: CONSTANT
        Information stored:
                -1

Label: v1t002
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t003
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t004
        Label attribute: CONSTANT
        Information stored:
                -1

Label: v1t005
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t006
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t007
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t008
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t009
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t010
        Label attribute: CONSTANT
        Length of information: 1
```

```
Label: v1t011
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t012
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t013
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t014
        Label attribute: CONSTANT
        Information stored:
                -1

Label: v1t015
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t016
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t017
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t018
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t019
        Label attribute: PARAMETER
        Information stored:
                1
**************************************************
```

```
======================
START card information
======================


=====================
Pseudo Code Generated
=====================


operator: ASSIGN
block number = 0
A←operand= v1t000
B←operand= v1t001

operator: ASSIGN
block number = 1
A←operand= v1t002
B←operand= v1t003

operator: ASSIGN
block number = 2
A←operand= func1
B←operand= v1t004

operator: ASSIGN
block number = 3
A←operand= var1
B←operand= v1t005

operator: ASSIGN
block number = 4
A←operand= v1t006
B←operand= v1t007

operator: ASSIGN
block number = 5
A←operand= v1t008
B←operand= func1

operator: NEG
block number = 6
A←operand= var1
B←operand= v1t010

operator: ASSIGN
block number = 6
A←operand= v1t009
B←operand= v1t010

operator: ASSIGN
block number = 12
A←operand= 1
B←operand= v1t014

operator: ASSIGN
block number = 14
A←operand= 1
B←operand= v1t016

operator: ASSIGN
block number = 16
A←operand= 1
B←operand= v1t018

Total number of errors = 13
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0         May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for DEPART statement
3               !
4                   simulate
5               a       variable        1
6               b       variable        1
7               stor1   storage         20
8               func1   function        r,d2
9               0.1,1/0.2,2
10              !
11      1               depart  s*stor1                 !not a good queue definition
                                  *
                >> Operand is not valid
12      2       label   depart                          !required operand missing
                           *
                >> Required operands missing
13      3               depart  1                       !ok
14      4               depart  a                       !A-operand not valid
                           *
                >> Operand is not valid
15      5               depart  a,                      !missing B-operand
                           *
                >> Operand is not valid
16      6               depart  ,                       !required operand missing
                           *
                >> Required operands missing
17      7               depart  a,b                     !illegal specification of SNA's
                           *
                >> Operand is not valid
18      8               depart  v*a,v*b                 !ok, variable specifications
19      9               depart  p1                      !illegal operand
                           *
                >> Operand is not valid
20              label   departr 1                       !multiple labels
                         *
                >> Multiply defined label
21      10              depart  1,v*b                   !ok
22      11              depart  v*a,2                   !ok
23      12              depart  s*1                     !not defined SNA
                           *
                >> Undefined operand
                           *
                >> Operand is not valid
24      13              depart  p*1,1                   !ok, correct specification
25                  end
```

```
**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: a
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: b
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: stor1
        Label attribute: STORAGE
        Length of information: 20

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label
        Label attribute: LABEL

Label: v1t000
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t001
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t002
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t003
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t004
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t005
        Label attribute: CONSTANT
        Information stored:
                1
**************************************************
```

```
======================
START card information
======================


======================
Pseudo Code Generated
======================


operator: DEPART
block number   2
A←operand= v1t000
B←operand= v1t001

operator: DEPART
block number   7
A←operand= a
B←operand= b

operator: DEPART
block number   9
A←operand= v1t002
B←operand= b

operator: DEPART
block number  10
A←operand= a
B←operand= v1t003

operator: DEPART
block number   12
A←operand= v1t004
B←operand= v1t005

Total number of errors - 10
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for END statement
3               !       FIRST of THREE test routines
4               !
5                   simulate
6                       end     a                       !no operands allowed
                          *
                    >> Operand is not valid
7                       endd                            !misspelling take as label
                          *
                    >> Label valid but no statement
8                       end                             !duplicate end
                        *
                    >> Multiple END statements
9               label   end                             !no label allowed
                            *
                    >> Label not allowed


************************************************

Symbol Table Contents after First Pass Completion
================================================

Label: label
        Label attribute: LABEL
************************************************

======================
START card information
======================



=====================
Pseudo Code Generated
=====================


Total number of errors - 4
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *

*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for END statement
3               !       SECOND of THREE test routines
4               !
5                   simulate
6                       end     ,b                      !syntax error due to comma
                          *
                    >> Operand is not valid
7               #$*     end                             !illegal label
                    *
                    >> Syntax error
8               .       end                             !syntax error
                      *
                    >> Syntax error



=====================
Pseudo Code Generated
=====================


Total number of errors - 3
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *

*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
```

```
1               !
2               !       parsing test for END statement
3               !       THIRD of THREE test routines
4                   simulate
5                   end                                             !ok


=====================
Pseudo Code Generated
=====================
```

```
*       GPSS V1.0         May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for ENTER statement
3               !
4                   simulate
5               a       variable        1
6               b       variable        1
7               stor1   storage         20
8               func1   function        r,d2
9               0.1,1/0.2,2
10              !
11        1             enter   s*stor1              !ok, storage label specification
12        2     label   enter                       !required operand missing
                                *
                >> Required operands missing
13        3             enter   1                   !ok
14        4             enter   a                   !A-operand not valid
                                *
                >> Operand is not valid
15        5             enter   a,                  !B-operand missing
                                *
                >> Operand is not valid
16        6             enter   ,                   !required operand missing
                                *
                >> Required operands missing
17        7             enter   a,b                 !illegal specification of SNA's
                                *
                >> Operand is not valid
18        8             enter   v*a,v*b             !ok, variable specifications
19        9             enter   p1                  !illegal operand
                                *
                >> Operand is not valid
20              label   enterr  1                   !multiple labels
                 *
                >> Multiply defined label
21        10            enter   1,v*b               !ok
22        11            enter   v*a,2               !ok
23        12            enter   s*1                 !not defined SNA
                                *
                >> Undefined operand
                                *
                >> Operand is not valid
24        13            enter   p*1,1               !ok, correct specification
25                  end
```

```
**********************************************

Symbol Table Contents after First Pass Completion
==============================================

Label: a
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: b
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: stor1
        Label attribute: STORAGE
        Length of information: 20

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: v1t000
        Label attribute: CONSTANT
        Information stored:
                1

Label: label
        Label attribute: LABEL

Label: v1t001
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t002
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t003
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t004
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t005
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t006
        Label attribute: CONSTANT
        Information stored:
                1
**********************************************
```

```
======================
START card information
======================



====================
Pseudo Code Generated
====================


operator: ENTER
block number - 0
A←operand= stor1
B←operand= v1t000

operator: ENTER
block number = 2
A←operand= v1t001
B←operand= v1t002

operator: ENTER
block number - 7
A←operand= a
B←operand= b

operator: ENTER
block number - 9
A←operand= v1t003
B←operand= b

operator: ENTER
block number - 10
A←operand= a
B←operand= v1t004

operator: ENTER
block number = 12
A←operand= v1t005
B←operand= v1t006

Total number of errors = 9
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for function statement
3               !
4                   simulate
5               func1   function        r1,d2           !incorrect specification of random number
                                    •
                >> Operand is not valid
6               .5,5/.8,8                               !error on line due to faulty function definitic
**n
                    •
                >> Syntax error
7               1func   function        r1.d2           !label error
                                •
                >> Illegal specification of label
8               .5,5/.8,8                               !error on line due to faulty function definitio
**n
                    •
                >> Syntax error
9               func2   functi          r,d2            !illegal function specification
                                •
                >> Syntax error
10              .5,5/.8,8                               !error on line due to faulty function definitio
**n
                    •
                >> Syntax error
11              func3   function        r,d             !illegal specification of discrete function
                                    •
                >> Illegal specification of numeric quantity
                                    •
                >> Illegal specification of operand
12              ..5,5/.8,8                              !error on line due to faulty function definiton
**
                    •
                >> Syntax error
13              func4   function        r,d*2           !illegal specification of discrete function
                                    •
                >> Illegal specification of numeric quantity
                                    •
                >> Illegal specification of operand
14              .5,5/.8,8                               !error on line due to faulty function definitio
**n
                    •
                >> Syntax error
15              func5   function        r,d2            !ok
16              .5,5/.8,8                               !ok
17              func6   function        r,d2            !ok
18              1,2/3,4                                 !illegal specification due to random number
                    •
                >> Syntax error
19              func7   function        r,d2            !ok
20              0.5,5/.6,6                              !ok
21              func8   function        r,d2            !ok
22              .5,5/.6.7                               !typing error
                        •
                >> Syntax error
23              func9   function        r,d3            !ok
24              .1,1/.2,3/.4,7                          !ok
25              func10  function        r,d3            !ok
26              .1,1/0.4,4/0.85,7                       !ok
```

```
**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: func1
        Label attribute: FUNCTION

Label: 1func
        Label attribute: LABEL

Label: func2
        Label attribute: LABEL

Label: func3
        Label attribute: FUNCTION

Label: func4
        Label attribute: FUNCTION

Label: func5
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.500000        5
                0.800000        8
        Length of information: 2

Label: func6
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.000000        0
                0.000000        0
        Length of information: 2

Label: func7
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.500000        5
                0.600000        6
        Length of information: 2

Label: func8
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.500000        5
                0.600000        0
        Length of information: 2

Label: func9
        Label attribute: FUNCTION
        Number of points in the function: 3
                0.100000        1
                0.200000        3
                0.400000        7
        Length of information: 3

Label: func10
        Label attribute: FUNCTION
        Number of points in the function: 3
                0.100000        1
                0.400000        4
                0.850000        7
        Length of information: 3
**************************************************
```

```
======================
START card information
======================



======================
Pseudo Code Generated
======================


>> No END statement encountered

Total number of errors   15
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0         May,1984
*     author: J. Van Dellon
1                 !
2                 !       parsing test for GATE statement
3                 !  .
4                     simulate
5                 1       storage         1
6                 var1    variable        1
7                 func1   function        r,d2
8                 0.1,1/0.2,2
9                 !
10        1       label1  gate u          s*1,label2              !invalid SNA
                                    *
                  >> Operand is not valid
11        2       label2  gate u          fn*func1,label3         !ok
12        3       label3  gate u          v*var1,label4           !ok
13        4       label4  gate u          p*1,label1              !ok
14        5       label5  gate u          1,label1                !ok
15        6               gate u          1                       !ok, shouldn't move until true
16        7               gate nu         s*1,label2              !invalid SNA
                                    *
                  >> Operand is not valid
17        8               gate nu         fn*func1,label2         !ok
18        9               gate nu         v*var1,label2           !ok
19        10              gate nu         p*1,label2              !ok
20        11              gate nu         1,label1                !ok
21        12              gate nu         1                       !ok, shouldn't move until true
22        13              gate sf         s*1,label1              !ok
23        14              gate sf         p*1,label1              !illegal SNA
                                   *
                  >> Operand is not valid
24        15              gate sf         v*var1,label1           !illegal SNA
                                    *
                  >> Operand is not valid
25        16              gate sf         fn*func1,label1         !illegal SNA
                                     *
                  >> Operand is not valid
26        17              gate sf         s*1                     !ok, shouldn't move until true
27        18              gate se         s*1,label1              !ok
28        19              gate se         p*1,label1              !illegal SNA
                                    *
                  >> Operand is not valid
29        20              gate se         fn*func1,label1         !illegal SNA
                                     *
                  >> Operand is not valid
30        21              gate se         v*var1,label1           !illegal SNA
                                    *
                  >> Operand is not valid
31        22              gate se         s*1                     !ok, shouldn't move until true
32        23              gate sf         1,1                     !illegal SNA
                                     *
                  >> Operand is not valid
33        24              gate sf         s*1,1                   !illegal label
                                    *
                  >> Illegal specification of label
34        25              gate snf        s*1,label1              !ok
35        26              gate snf        p*1,label1              !illegal SNA
                                    *
                  >> Operand is not valid
36        27              gate snf        v*var1,label1           !illegal SNA
                                    *
                  >> Operand is not valid
37        28              gate snf        fn*func1,label1         !illegal SNA
                                     *
                  >> Operand is not valid
38        29              gate sne        s*1,label1              !ok
39        30              gate sne        p*1,label1              !illegal SNA
                                    *
                  >> Operand is not valid
40        31              gate sne        fn*func1,label1         !illegal SNA
                                     *
                  >> Operand is not valid
41        32              gate sne        v*var1,label1           !illegal SNA
                                    *
                  >> Operand is not valid
```

```
42      33              gate snf        1,1                     !illegal SNA
                                *
                >> Operand is not valid
43      34              gate snf        s*1,1                   !illegal label
                                *
                >> Illegal specification of label
44      35              gate ru         1,label1                !illegal mneumonic
                                *
                >> Illegal option specified
45              end
```

```
**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label1
        Label attribute: LABEL

Label: label2
        Label attribute: LABEL

Label: label3
        Label attribute: LABEL

Label: label4
        Label attribute: LABEL

Label: v1t000
        Label attribute: PARAMETER
        Information stored:
                1

Label: label5
        Label attribute: LABEL

Label: v1t001
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t002
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t003
        Label attribute: LABEL
        Length of information: 1

Label: v1t004
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t005
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t006
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t007
        Label attribute: LABEL
        Length of information: 1
```

```
Label: v1t008
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t009
        Label attribute: LABEL
        Length of information: 1

Label: v1t010
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t011
        Label attribute: LABEL
        Length of information: 1

Label: v1t012
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t013
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t014
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t015
        Label attribute: CONSTANT
        Information stored:
                1
**************************************************
```

```
======================
START card information
======================


======================
Pseudo Code Generated
======================


operator: U
block number   1
A←operand= func1
B←operand= * NULL *

operator: BNE
block number - 1
A←operand= label3
B←operand= * NULL *

operator: U
block number - 2
A←operand= var1
B←operand= * NULL *

operator: BNE
block number = 2
A←operand= label4
B←operand= * NULL *

operator: U
block number - 3
A←operand= v1t000
B←operand= * NULL *

operator: BNE
block number - 3
A←operand= label1
B←operand= * NULL *

operator: U
block number - 4
A←operand= v1t001
B←operand= * NULL *

operator: BNE
block number - 4
A←operand= label1
B←operand= * NULL *

operator: U
block number - 5
A←operand= v1t002
B←operand= * NULL *

operator: BNE
block number - 5
A←operand= v1t003
B←operand= * NULL *

operator: NU
block number = 7
A←operand= func1
B←operand= * NULL *

operator: BNE
block number = 7
A←operand= label2
B←operand= * NULL *

operator: NU
block number = 8
A←operand= var1
B←operand= * NULL *
```

```
operator: BNE
block number - 8
A←operand= label2
B←operand= * NULL *

operator: NU
block number = 9
A←operand= v1t004
B←operand= * NULL *

operator: BNE
block number - 9
A←operand= label2
B←operand= * NULL *

operator: NU
block number   10
A←operand= v1t005
B←operand= * NULL *

operator: BNE
block number = 10
A←operand= label1
B←operand= * NULL *

operator: NU
block number - 11
A←operand= v1t006
B←operand= * NULL *

operator: BNE
block number - 11
A←operand= v1t007
B←operand= * NULL *

operator: SF
block number - 12
A←operand= 1
B←operand= * NULL *

operator: BNE
block number - 12
A←operand= label1
B←operand= * NULL *

operator: SF
block number   16
A←operand= 1
B←operand= * NULL *

operator: BNE
block number - 16
A←operand= v1t009
B←operand= * NULL *

operator: SE
block number - 17
A←operand= 1
B←operand= * NULL *

operator: BNE
block number - 17
A←operand= label1
B←operand= * NULL *

operator: SE
block number - 21
A←operand= 1
B←operand= * NULL *

operator: BNE
block number - 21
A←operand= v1t011
B←operand= * NULL *
```

```
operator: SF
block number - 23
A←operand= 1
B←operand= * NULL *

operator: SNF
block number - 24
A←operand= 1
B←operand= * NULL *

operator: BNE
block number   24
A←operand= label1
B←operand= * NULL *

operator: SNE
block number - 28
A←operand= 1
B←operand= * NULL *

operator: BNE
block number - 28
A←operand= label1
B←operand= * NULL *

operator: SNF
block number = 33
A←operand= 1
B←operand= * NULL *

Total number of errors = 19
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for GENERATE statement
3               !
4                   simulate
5               1       storage 1
6               var1    variable        1
7               func1   function        r,d2
8               0.1,1/0.2,2
9               !
10      1           generate    1               !ok
11      2           generate    5,1             !ok
12      3           generate    1,p*1           !can't cause no parameters are evaluated yet
                                    *
                >> Operand is not valid
13      4           generate    1,fn*func1      !ok
14      5           generate    1,v*var1        !ok
15      6           generate    p*1,1           !can't cause no parameters are evaluated yet
                                  *
                >> Operand is not valid

16      7           generate    v*var1,1        !ok
17      8           generate    fn*func1,1      !ok
18      9           generate    s*1,1           !illegal SNA
                                  *
                >> Operand is not valid
19      10          generate    1,s*1           !illegal SNA
                                    *
                >> Operand is not valid
20      11          generate    ,1              !required operand missing
                                 *
                >> Required operands missing
21      12          generate    1,              !syntax error
                                    *
                >> Syntax error
22      13          generate    fn*func2        !undefined function
                                      *
                >> Undefined operand
                                     *
                >> Operand is not valid
23      14          generate    1       1       !syntax error
                                   *
                >> Syntax error
24      15          generate    1,1,1           !syntax error
                                    *
                >> Syntax error
26              end
```

```
**********************************************

Symbol Table Contents after First Pass Completion
================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: v1t000
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t001
        Label attribute: CONSTANT
        Information stored:
                5

Label: v1t002
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t003
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t004
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t005
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t006
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t007
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t008
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t009
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t010
        Label attribute: CONSTANT
        Information stored:
```

```
                1

Label: v1t011
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t012
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t013
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t014
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t015
        Label attribute: CONSTANT
        Information stored:
                1
************************************************
```

```
======================
START card information
======================


=====================
Pseudo Code Generated
=====================


operator: GENERATE
block number - 0
A←operand= v1t000
B←operand= * NULL *

operator: GENERATE
block number - 1
A←operand= v1t001
B←operand= v1t002

operator: GENERATE
block number   2
A←operand= v1t003
B←operand= v1t004

operator: GENERATE
block number   3
A←operand= v1t005
B←operand= func1

operator: GENERATE
block number - 4
A←operand= v1t006
B←operand= var1

operator: GENERATE
block number - 5
A←operand= v1t007
B←operand= v1t008

operator: GENERATE
block number   6
A←operand= var1
B←operand= v1t009

operator: GENERATE
block number   7
A←operand= func1
B←operand= v1t010

Total number of errors - 10
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
•       GPSS V1.0          May,1984
•     author: J. Van Dellon
1               !
2               !       parsing test for LEAVE statement
3               !
4                   simulate
5               a       variable        1
6               b       variable        1
7               stor1   storage         20
8               func1   function        r,d2
9               0.1,1/0.2,2
10              !
11      1               leave   s*stor1                 !ok, storage label specification
12      2       label   leave                           !required operand missing
                                •
                >> Required operands missing
13      3               leave   1                       !ok
14      4               leave   a                       !illegal A-operand
                                •
                >> Operand is not valid
15      5               leave   a,                      !no B-operand
                                •
                >> Operand is not valid
16      6               leave   ,                       !required operand missing
                                •
                >> Required operands missing
17      7               leave   a,b                     !illegal specification of SNA's
                                •
                >> Operand is not valid
18      8               leave   v*a,v*b                 !ok, variable specifications
19      9               leave   p1                      !illegal operand
                                 •
               ,>> Operand is not valid
20              label   leaver  1                       !multiple labels
                 •
                >> Multiply defined label
21      10              leave   1,v*b                   !ok
22      11              leave   v*a,2                   !ok
23      12              leave   s*1                     !not defined SNA
                                  •
                >> Undefined operand
                                  •
                >> Operand is not valid
24      13              leave   p*1,1                   !ok, correct specification
25                  end
```

```
**********************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: a
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: b
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: stor1
        Label attribute: STORAGE
        Length of information: 20

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: v1t000
        Label attribute: CONSTANT
        Information stored:
                1

Label: label
        Label attribute: LABEL

Label: v1t001
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t002
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t003
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t004
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t005
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t006
        Label attribute: CONSTANT
        Information stored:
                1
**********************************************
```

```
======================
START card information
======================



=====================
Pseudo Code Generated
=====================


operator: LEAVE
block number   0
A←operand= stor1
B←operand= v1t000

operator: LEAVE
block number   2
A←operand= v1t001
B←operand= v1t002

operator: LEAVE
block number - 7
A←operand= a
B←operand= b

operator: LEAVE
block number - 9
A←operand= v1t003
B←operand= b

operator: LEAVE
block number   10
A←operand= a
B←operand= v1t004

operator: LEAVE
block number   12
A←operand= v1t005
B←operand= v1t006

Total number of errors - 9
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0          May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for QUEUE statement
3               !
4                   simulate
5               a       variable        1
6               b       variable        1
7               stor1   storage         20
8               func1   function        r,d2
9               0.1,1/0.2,2
10              !
11      1               queue   s*stor1                 !ok, storage label specification
                                    *
                >> Operand is not valid
12      2       label   queue                           !required operand missing
                              *
                >> Required operands missing
13      3               queue   1                       !ok
14      4               queue   a                       !illegal A-operand
                                *
                >> Operand is not valid
15      5               queue   a,                      !no B-operand
                                *
                >> Operand is not valid
16      6               queue   ,                       !required operand missing
                                *
                >> Required operands missing
17      7               queue   a,b                     !illegal specification of SNA's
                                *
                >> Operand is not valid
18      8               queue   v*a,v*b                 !ok, variable specifications
19      9               queue   p1                      !illegal operand
                                *
                >> Operand is not valid
20              label   queuer  1                       !multiple labels
                        *
                >> Multiply defined label
21      10              queue   1,v*b                   !ok
22      11              queue   v*a,2                   !ok
23      12              queue   s*1                     !not defined SNA
                                *
                >> Undefined operand
                                *
                >> Operand is not valid
24      13              queue   p*1,1                   !ok, correct specification
25                  end
```

```
**********************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: a
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: b
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: stor1
        Label attribute: STORAGE
        Length of information: 20

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label
        Label attribute: LABEL

Label: v1t000
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t001
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t002
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t003
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t004
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t005
        Label attribute: CONSTANT
        Information stored:
                1
**********************************************
```

```
======================
START card information
======================


=====================
Pseudo Code Generated
=====================


operator: QUEUE
block number - 2
A←operand= v1t000
B←operand= v1t001

operator: QUEUE
block number - 7
A←operand= a
B←operand= b

operator: QUEUE
block number - 9
A←operand= v1t002
B←operand= b

operator: QUEUE
block number - 10
A←operand= a
B←operand= v1t003

operator: QUEUE
block number   12
A←operand= v1t004
B←operand= v1t005

Total number of errors = 10
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for RELEASE statement
3               !
4                   simulate
5               1       storage 1
6               jacky   variable        1
7               func1   function r,d2
8               0.1,1/0.2,2
9               !
10      1       label   release                         !required operand missing
                                  *
                >> Required operands missing
11      2       1a      release 2                       !illegal label
                          *
                >> Illegal specification of label
12      3               release 1                       !ok
13      4               release a                       !not allowed SNA
                          *
                >> Operand is not valid
14      5               release p1                      !illegal specification of SNA
                          *
                >> Operand is not valid
15      6               release 1,                      !syntax error
                          *
                >> Syntax error
16      7               release ,1                      !required operand missing
                          *
                >> Required operands missing
17      8               release P1                      !illegal specification of SNA
                          *
                >> Operand is not valid
18      9               release p*1                     !ok, parameter
19      10              release P*1                     !ok, parameter
20      11              release s*1                     !not allowable SNA
                                  *
                >> Operand is not valid
21      12              release S*1                     !not allowable SNA
                                  *
                >> Operand is not valid
22      13              release v*jacky                 !ok, variable
23      14              release V*jacky                 !ok, variable
24      15              release f*1                     !facility not allowed
                                  *
                >> Operand is not valid
25      16              release FN*func1                !ok, function
26      17              release fn*func1                !ok, function
27      18              release p*1                     !ok, parameter
28      19              release P*1                     !ok, parameter
29      20              release 1       1               !syntax error
                                  *
                >> Syntax error
30              .       release                         !syntax error
                  *
                >> Syntax error
31              @##     release                         !illegal label
                  *
                >> Syntax error
32                  end
```

```
**********************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: jacky
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label
        Label attribute: LABEL

Label: 1a
        Label attribute: LABEL

Label: v1t000
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t001
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t002
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t003
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t004
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t005
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t006
        Label attribute: CONSTANT
        Information stored:
                1
**********************************************
```

```
======================
START card information
======================


====================
Pseudo Code Generated
====================


operator: RELEASE
block number   2
A←operand= v1t000
B←operand= * NULL *

operator: RELEASE
block number - 8
A←operand= v1t002
B←operand= * NULL *

operator: RELEASE
block number   9
A←operand= v1t003
B←operand= * NULL *

operator: RELEASE
block number   12
A←operand= jacky
B←operand= * NULL *

operator: RELEASE
block number   13
A←operand= jacky
B←operand= * NULL *

operator: RELEASE
block number   15
A←operand= func1
B←operand= * NULL *

operator: RELEASE
block number - 16
A←operand= func1
B←operand= * NULL *

operator: RELEASE
block number   17
A←operand= v1t004
B←operand= * NULL *

operator: RELEASE
block number   18
A←operand= v1t005
B←operand= * NULL *

Total number of errors - 13
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0         May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for SEIZE statement
3               !
4                   simulate
5               1       storage 1
6               jacky   variable        1
7               func1   function r,d2
8               0.1,1/0.2,2
9               !
10       1      label   seize                           !required operand missing
                                 *
                >> Required operands missing
11       2      1a      seize   2                       !illegal label
                         *
                >> Illegal specification of label
12       3              seize   1                       !ok
13       4              seize   a                       !not allowed SNA
                         *
                >> Operand is not valid
14       5              seize   p1                      !illegal specification of SNA
                         *
                >> Operand is not valid
15       6              seize   1,                      !syntax error
                         *
                >> Syntax error
16       7              seize   ,1                      !required operand missing
                         *
                >> Required operands missing
17       8              seize   P1                      !illegal specification of SNA
                         *
                >> Operand is not valid
18       9              seize   p*1                     !ok, parameter
19       10             seize   P*1                     !ok, parameter
20       11             seize   s*1                     !not allowable SNA
                           *
                >> Operand is not valid
21       12             seize   S*1                     !not allowable SNA
                           *
                >> Operand is not valid
22       13             seize   v*jacky                 !ok, variable
23       14             seize   V*jacky                 !ok, variable
24       15             seize   f*1                     !facility not allowed
                           *
                >> Operand is not valid
25       16             seize   FN*func1                !ok, function
26       17             seize   fn*func1                !ok, function
27       18             seize   p*1                     !ok, parameter
28       19             seize   P*1                     !ok, parameter
29       20             seize   1       1               !syntax error
                           *
                >> Syntax error
30              ,       seize                           !syntax error
                  *
                >> Syntax error
31              @##     seize                           !illegal label
                  *
                >> Syntax error
32                  end
```

```
**********************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: jacky
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label
        Label attribute: LABEL

Label: 1a
        Label attribute: LABEL

Label: v1t000
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t001
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t002
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t003
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t004
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t005
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t006
        Label attribute: CONSTANT
        Information stored:
                1
**********************************************
```

```
======================
START card information
======================


======================
Pseudo Code Generated
======================


operator: SEIZE
block number   2
A←operand= v1t000
B←operand= * NULL *

operator: SEIZE
block number - 8
A←operand= v1t002
B←operand= * NULL *

operator: SEIZE
block number - 9
A←operand= v1t003
B←operand= * NULL *

operator: SEIZE
block number = 12
A←operand= jacky
B←operand= * NULL *

operator: SEIZE
block number = 13
A←operand= jacky
B←operand= * NULL *

operator: SEIZE
block number - 15
A←operand= func1
B←operand= * NULL *

operator: SEIZE
block number = 16
A←operand= func1
B←operand= * NULL *

operator: SEIZE
block number - 17
A←operand= v1t004
B←operand= * NULL *

operator: SEIZE
block number = 18
A←operand= v1t005
B←operand= * NULL *

Total number of errors - 13
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for SIMULATE statement
3               !       FIRST of THREE test routines
4               !
5                       simulate        a                       !no operands allowed
                                  *
                >> Operand is not valid
6                       simulatte                               !misspelling take as label
                                  *
                >> Label valid but no statement
7                       simulate                                !duplicate simulate
                                  *
                >> Multiple SIMULATION statements
8               label   simulate                                !no label allowed
                                  *
                >> Label not allowed
9                   end

**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: label
        Label attribute: LABEL
**************************************************

======================
START card information
======================



======================
Pseudo Code Generated
======================


Total number of errors - 4
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *

*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for SIMULATE statement
3               !       SECOND of THREE test routines
4               !
5                       simulate        ,b                      !syntax error due to comma
                                  *
                >> Syntax error
6               #$*     simulate                                !illegal label
                    *
                >> Syntax error
7               ,       simulate                                !syntax error
                      *
                >> Syntax error
8                   end



======================
Pseudo Code Generated
======================


Total number of errors - 3
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *

*       GPSS V1.0         May,1984
*     author: J. Van Dellon
```

```
1               !
2               !       parsing test for SIMULATE statement
3               !       THIRD of THREE test routines
4               !
5                   simulate                                    !ok
6                   end
```

```
====================
Pseudo Code Generated
====================
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for START statement
3               !
4                   simulate
5               !
6               label   start   1                               !label not allowed
                        *
                >> Label not allowed
7               .       start                                   !syntax error
                  *
                >> Syntax error
8                       start   1                               !ok
9                       start   a                               !illegal operand
                        *
                >> Undefined operand
10                      start   ,NP,3                           !required operand missing
                        *
                >> Required operands missing
11                      start   1,np                            !ok
12                      start   1,,                             !syntax correct, just no arguements
13                      start   1,,c                            !illegal operand
                          *
                >> Undefined operand
14                      start   1,,2                            !ok
15                      start   1,NULL                          !ok
16                      start   1,bb                            !illegal option
                          *
                >> Operand is not valid
17                      start   1       np      1               !syntax error
                          *
                >> Syntax error
18                  end
```

```
**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: label
       Label attribute: LABEL
**************************************************

**********************
START card information
**********************

START card #1:  simulation time 1
                NO SNAP interval printout requested

START card #2:  simulation time 1
                SNAP interval printout requested
                Interval printout time is 2 units


**********************
Pseudo Code Generated
**********************


Total number of errors = 7
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for STORAGE statement
3               !
4                   simulate
5               !
6                       storage                         !missing label
                                •
                >> Label required
7               a       storage 2                       !ok
8               1       storage 3                       !ok
9               1a      storage 4                       !ok
                                •
                >> Illegal specification of numeric quantity
                                •
                >> Illegal specification of label
10              1       storage 5                       !duplicate label
                    •
                >> Multiply defined label
11              2       storage                         !required operand missing
                                •
                >> Required operands missing
12              3       storage six                     !illegal operand
                                •
                >> Operand is not valid
13              4       storage 7,                      !syntax error
                                •
                >> Syntax error
14              5       storage 8,9                     !syntax error
                                •
                >> Syntax error
15              6       storage 10      11              !syntax error
                                    •
                >> Syntax error
16              .       storage                         !syntax error
                    •
                >> Syntax error
17              $%↑↑    storage 1                       !illegal label
                    •
                >> Syntax error
18                  end
```

```
**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: a
        Label attribute: STORAGE
        Length of information: 2

Label: 1
        Label attribute: STORAGE
        Length of information: 3

Label: 1a
        Label attribute: STORAGE

Label: 2
        Label attribute: STORAGE

Label: 3
        Label attribute: STORAGE

Label: 4
        Label attribute: STORAGE
        Length of information: 7

Label: 5
        Label attribute: STORAGE
        Length of information: 8

Label: 6
        Label attribute: STORAGE
        Length of information: 10
**************************************************
```

```
======================
START card information
======================


=====================
Pseudo Code Generated
=====================


Total number of errors - 11
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for TERMINATE statement
3               !
4                   simulate
5               1       storage 1
6               var1    variable        1
7               func1   function        r,d2
8               0.1,1/0.2,2
9               !
10      1       label   terminate                               !ok
11      2               terminate                               !ok, default 1
12      3               terminate       1                       !ok
13      4               terminate       ,                       !syntax error
                          *
                >> Syntax error
14      5               terminate       a                       !illegal operand
                          *
                >> Operand is not valid
15      6               terminate       2,                      !syntax error
                            *
                >> Syntax error
16      7               terminate       3       4               !syntax error
                              *
                >> Syntax error
17      8               terminate       5                       !ok
18                      terminatte                              !misspelling, error
                            *
                >> Label valid but no statement
19              ,       terminate                               !syntax error
                  *
                >> Syntax error
20              @##     terminate                               !illegal label
                  *
                >> Syntax error
21      9               terminate       P1                      !illegal operand
                          *
                >> Operand is not valid
22      10              terminate       p1                      !illegal operand
                            *
                >> Operand is not valid
23      11              terminate       p*1                     !no SNA's allowed
                              *
                >> Operand is not valid
24      12              terminate       P*1                     !no SNA's allowed
                              *
                >> Operand is not valid
25      13              terminate       s*1                     !no SNA's allowed
                              *
                >> Operand is not valid
26      14              terminate       S*1                     !no SNA's allowed
                              *
                >> Operand is not valid
27      15              terminate       v*1                     !no SNA's allowed
                              *
                >> Operand is not valid
28      16              terminate       V*1                     !no SNA's allowed
                              *
                >> Operand is not valid
29      17              terminate       fn*1                    !no SNA's allowed
                               *
                >> Operand is not valid
30      18              terminate       FN*1                    !no SNA's allowed
                              *
                >> Operand is not valid
31                  end
```

```
**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label
        Label attribute: LABEL

Label: v1t000
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t001
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t002
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t003
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t004
        Label attribute: CONSTANT
        Information stored:
                3

Label: v1t005
        Label attribute: CONSTANT
        Information stored:
                5
**************************************************
```

```
======================
START card information
======================



=====================
Pseudo Code Generated
=====================


operator: TERMINATE
block number = 0
A←operand= v1t000
B←operand= * NULL *

operator: TERMINATE
block number - 1
A←operand= v1t001
B←operand= * NULL *

operator: TERMINATE
block number = 2
A←operand= v1t002
B←operand= * NULL *

operator: TERMINATE
block number = 7
A←operand= v1t005
B←operand= * NULL *

Total number of errors = 17
Simulation phase will not occur due to ERRORS
*       GPSS exiting     *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for TEST statement
3               !
4                       simulate
5               1       storage         1
6               var1    variable        1
7               func1   function        r,d2
8               0.1,1/0.2,2
9               !
10      1       label1  test e  p*1,p*2,label2          !ok
11      2       label2  test e  v*var1,p*1,label3       !ok
12      3       label3  test e  fn*func1,p*1,label1     !ok
13      4               test e  1,p*1,label2            !ok
14      5               test e  1,2,label3              !ok
15      6               test e  1,p*1,label1            !ok
16      7               test e  1,fn*func1,label2       !ok
17      8               test e  1,v*var1,label3         !ok
18      9               test l  1,2,label3              !ok
19      10              test g  1,2,label3              !ok
20      11              test ge 1,2,label3              !illegal option
                                *
                ?? Illegal option specified
21      12              test l  s*1,1,label3            !illegal SNA
                                  *
                ?? Operand is not valid
22      13              test g  1,s*1,label3            !illegal SNA
                                    *
                ?? Operand is not valid
23      14              test e  ,1,label3               !required operand missing
                                *
                ?? Required operands missing
24      15              test l  1,,label3               !required operand missing
                                  *
                ?? Required operands missing
25      16              test g  1,1                     !ok
26      17              test e  1,1,label4              !incorrect label error second pass
27      18              test l  a,1,label1              !illegal operand
                                *
                ?? Operand is not valid
28      19              test g  1,a,label1              !illegal operand
                                  *
                ?? Operand is not valid
29      20              test e  v*var2,p*2,label2       !undefined operand
                                    *
                ?? Undefined operand
                                     *
                ?? Operand is not valid
30                      end
```

```
**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label1
        Label attribute: LABEL

Label: v1t000
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t001
        Label attribute: PARAMETER
        Information stored:
                2

Label: label2
        Label attribute: LABEL

Label: v1t002
        Label attribute: PARAMETER
        Information stored:
                1

Label: label3
        Label attribute: LABEL

Label: v1t003
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t004
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t005
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t006
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t007
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t00B
        Label attribute: CONSTANT
        Information stored:
                1
```

```
Label: v1t009
        Label attribute: PARAMETER
        Information stored:
                1

Label: v1t010
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t011
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t012
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t013
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t014
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t015
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t016
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t017
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t018
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t019
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t020
        Label attribute: LABEL
        Length of information: 1

Label: v1t021
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t022
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t023
        Label attribute: CONSTANT
        Information stored:
                1
**************************************************
```

```
=====================
START card information
=====================
>>> Undefined label label4


====================
Pseudo Code Generated
====================


operator: EQ
block number = 0
A←operand= v1t000
B←operand= v1t001

operator: BNE
block number = 0
A←operand= label2
B←operand= * NULL *

operator: EQ
block number = 1
A←operand= var1
B←operand= v1t002

operator: BNE
block number - 1
A←operand= label3
B←operand= * NULL *

operator: EQ
block number ='2
A←operand= func1
B←operand= v1t003

operator: BNE
block number = 2
A←operand= label1
B←operand= * NULL *

operator: EQ
block number = 3
A←operand= v1t004
B←operand= v1t005

operator: BNE
block number = 3
A←operand= label2
B←operand= * NULL *

operator: EQ
block number = 4
A←operand= v1t006
B←operand= v1t007

operator: BNE
block number = 4
A←operand= label3
B←operand= * NULL *

operator: EQ
block number = 5
A←operand= v1t008
B←operand= v1t009

operator: BNE
block number = 5
A←operand= label1
B←operand= * NULL *

operator: EQ
block number = 6
A←operand= v1t010
```

```
B←operand= func1

operator: BNE
block number = 6
A←operand= label2
B←operand= * NULL *

operator: EQ
block number = 7
A←operand= v1t011
B←operand= var1

operator: BNE
block number = 7
A←operand= label3
B←operand= * NULL *

operator: LT
block number = 8
A←operand= v1t012
B←operand= v1t013

operator: BNE
block number = 8
A←operand= label3
B←operand= * NULL *

operator: GT
block number = 9
A←operand= v1t014
B←operand= v1t015

operator: BNE
block number = 9
A←operand= label3
B←operand= * NULL *

operator: GT
block number = 15
A←operand= v1t01B
B←operand= v1t019

operator: BE
block number = 15
A←operand= v1t020
B←operand= * NULL *

operator: EQ
block number = 16
A←operand= v1t021
B←operand= v1t022

operator: BNE
block number = 16
A←operand= label4
B←operand= * NULL *

Total number of errors = 10
Simulation phase will not occur due to ERRORS
*       GPSS exiting    *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       Purpose:
3               !               The following program is used for a generic debugging tool
4               !       for the GPSS compiler/simulator thesis.  The following program
5               !       depicts a traffic problem over a bridge where one lane has been shut
6               !       off and the traffic is redirected through the other lane by means of
7               !       a traffic light at each end of the bridge.
8               !
9               !       Written:
10              !               Jacky Van Dellon        March 15, 1984
11              !
12              !-------------------------------------------------------------------------
13              !
14              !       Parameters:
15              !               4       - keep count of cars in direction 2
16              !               5       - keep count of cars in direction 1
17              !
18                  simulate
19              !
20              !       Traffic lights problem
21              !       ======================
22              expon   function        r,d23
23              0.0,0/0.1,104/0.2,222/0.3,355/0.4,509/0.5,690/0.6,915/0.7,1200/0.8,1600/
24              0.84,1830/0.88,2120/0.9,2300/0.92,2520/0.94,2810/0.95,2990/0.96,3200/0.97,3500/
25              0.98,3900/0.99,4600/0.995,5300/0.998,6200/0.999,7000/0.9997,8000
27              gre1    variable        600     ! length of time for green light in direction 1
28              gre2    variable        450     ! length of time for green light in direction 2
29              stop    storage         1       ! storage to restrict flow of transactions
30              1       storage         30      ! storage used for entering cars in direction 1
31              2       storage         30      ! storage used for entering cars in direction 2
32              !
33              !       traffic from direction 1
34              !       ========================
35      1               generate 90,fn*expon            !generate cars
36      2               queue   1                       !enter queue
37      3               seize   1                       !seize starting place
38      4               gate u  1                       !is light green
39      5               transfer        ,delay          !null transfer
40              !
41      6       null1   advance 100                     !null block
42              !
43      7       delay   advance 20                      !start car
44      8               assign  5,+1                    !keep count of cars
45      9               enter   1,1                     !enter storage
46      10              depart  1                       !leave queue
47      11              release 1                       !release starting place
48      12              leave   1                       !leave storage
49      13              terminate                       !car is over bridge
50              !
51              !       traffic from direction 2
52              !       ========================
53      14              generate 120,fn*expon           !generate cars
54      15              queue   2                       !enter queue
55      16              seize   2                       !seize starting position
56      17              gate u  2                       !is light 2 green
57      18              test g  p*5,10,around           !check for ten cars in the counter
58              !
59      19      null2   advance 100                     !null block
60              !
61      20      around  advance 20                      !start car
62      21              assign  4,+1                    !count cars in direction 2
63      22              depart  2                       !leave starting place
64      23              release 2                       !release starting place
65      24              terminate                       !car is over bridge
66              !
67              !       traffic lights
68              !       ==============
69      25              generate 1                      !just start this sequence up
70      26              gate se s*stop                  !make sure only one goes through
71      27              enter   s*stop                  !one is through and starts cycle
72              !
73      28      loop    advance 550                     !both lights red
74      29              seize   1                       !light 1 becomes green
```

```
75      30              advance v*gre1                  !green time for 1
76      31              release 1                       !light one turns red
77      32              advance 550                     !both lights red
78      33              seize   2                       !light 2 becomes green
79      34              advance v*gre2                  !green time for 2
80      35              release 2                       !light two turns red
81      36              transfer ,loop                  !begin new light cycle
82              !
83              !       auxiliary clock
84              !       ===============
85      37              generate 36000                  !one transaction per hour
86      38              terminate 1                     !decrement
87              !
88              !       start
89              !       =====
90                  start               1
91              !
92                  end
```

```
**********************************************

Symbol Table Contents after First Pass Completion
=================================================

Label: expon
        Label attribute: FUNCTION
        Number of points in the function: 23
                0.000000        0
                0.100000        104
                0.200000        222
                0.300000        355
                0.400000        509
                0.500000        690
                0.600000        915
                0.700000        1200
                0.800000        1600
                0.840000        1830
                0.880000        2120
                0.900000        2300
                0.920000        2520
                0.940000        2810
                0.950000        2990
                0.960000        3200
                0.970000        3500
                0.980000        3900
                0.990000        4600
                0.995000        5300
                0.998000        6200
                0.999000        7000
                0.999700        8000
        Length of information: 23

Label: gre1
        Label attribute: VARIABLE
        Length of information: 4
        Information stored:
                600:

Label: gre2
        Label attribute: VARIABLE
        Length of information: 4
        Information stored:
                450:

Label: stop
        Label attribute: STORAGE
        Length of information: 1

Label: 1
        Label attribute: STORAGE
        Length of information: 30

Label: 2
        Label attribute: STORAGE
        Length of information: 30

Label: v1t000
        Label attribute: CONSTANT
        Information stored:
                90

Label: v1t001
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t002
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t003
        Label attribute: CONSTANT
        Information stored:
```

```
                1

Label: v1t004
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t005
        Label attribute: LABEL
        Length of information: 1

Label: null1
        Label attribute: LABEL

Label: v1t006
        Label attribute: CONSTANT
        Information stored:
                100

Label: delay
        Label attribute: LABEL

Label: v1t007
        Label attribute: CONSTANT
        Information stored:
                20

Label: v1t008
        Label attribute: PARAMETER
        Information stored:
                5

Label: v1t009
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t010
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t011
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t012
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t013
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t014
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t015
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t016
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t017
        Label attribute: CONSTANT
        Information stored:
                1
```

```
Label: v1t018
        Label attribute: CONSTANT
        Information stored:
                120

Label: v1t019
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t020
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t021
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t022
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t023
        Label attribute: LABEL
        Length of information: 1

Label: v1t024
        Label attribute: PARAMETER
        Information stored:
                5

Label: v1t025
        Label attribute: CONSTANT
        Information stored:
                10

Label: null2
        Label attribute: LABEL

Label: v1t026
        Label attribute: CONSTANT
        Information stored:
                100

Label: around
        Label attribute: LABEL

Label: v1t027
        Label attribute: CONSTANT
        Information stored:
                20

Label: v1t028
        Label attribute: PARAMETER
        Information stored:
                4

Label: v1t029
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t030
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t031
        Label attribute: CONSTANT
        Information stored:
                1
```

```
Label: v1t032
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t033
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t034
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t035
        Label attribute: LABEL
        Length of information: 1

Label: v1t036
        Label attribute: CONSTANT
        Information stored:
                1

Label: loop
        Label attribute: LABEL

Label: v1t037
        Label attribute: CONSTANT
        Information stored:
                550

Label: v1t038
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t039
        Label attribute: CONSTANT
        Information stored:
                1

Label: v1t040
        Label attribute: CONSTANT
        Information stored:
                550

Label: v1t041
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t042
        Label attribute: CONSTANT
        Information stored:
                2

Label: v1t043
        Label attribute: CONSTANT
        Information stored:
                36000

Label: v1t044
        Label attribute: CONSTANT
        Information stored:
                1
*************************************************
```

```
**********
START card information
**********

START card #1:  simulation time 1
                NO SNAP interval printout requested


**********
Pseudo Code Generated
**********


operator: GENERATE
block number = 0
A-operand= v1t000
B-operand= expon

operator: QUEUE
block number = 1
A-operand= v1t001
B-operand= v1t002

operator: SEIZE
block number = 2
A-operand= v1t003
B-operand= * NULL *

operator: U
block number = 3
A-operand= v1t004
B-operand= * NULL *

operator: BNE
block number = 3
A-operand= v1t005
B-operand= * NULL *

operator: BR
block number = 4
A-operand= * NULL *
B-operand= delay

operator: ADVANCE
block number = 5
A-operand= v1t006
B-operand= * NULL *

operator: ADVANCE
block number = 6
A-operand= v1t007
B-operand= * NULL *

operator: ASSIGN
block number = 7
A-operand= v1t008
B-operand= v1t009

operator: ENTER
block number = 8
A-operand= v1t010
B-operand= v1t011

operator: DEPART
block number = 9
A-operand= v1t012
B-operand= v1t013

operator: RELEASE
block number = 10
A-operand= v1t014
B-operand= * NULL *

operator: LEAVE
```

```
block number = 11
A←operand= v1t015
B←operand= v1t016

operator: TERMINATE
block number = 12
A←operand= v1t017
B←operand= * NULL *

operator: GENERATE
block number = 13
A←operand= v1t018
B←operand= expon

operator: QUEUE
block number = 14
A←operand= v1t019
B←operand= v1t020

operator: SEIZE
block number = 15
A←operand= v1t021
B←operand= * NULL *

operator: U
block number = 16
A←operand= v1t022
B←operand= * NULL *

operator: BNE
block number = 16
A←operand= v1t023
B←operand= * NULL *

operator: GT
block number = 17
A←operand= v1t024
B←operand= v1t025

operator: BNE
block number = 17
A←operand= around
B←operand= * NULL *

operator: ADVANCE
block number = 18
A←operand= v1t026
B←operand= * NULL *

operator: ADVANCE
block number = 19
A←operand= v1t027
B←operand= * NULL *

operator: ASSIGN
block number = 20
A←operand= v1t028
B←operand= v1t029

operator: DEPART
block number = 21
A←operand= v1t030
B←operand= v1t031

operator: RELEASE
block number = 22
A←operand= v1t032
B←operand= * NULL *

operator: TERMINATE
block number = 23
A←operand= v1t033
B←operand= * NULL *

operator: GENERATE
block number = 24
```

```
A←operand= v1t034
B←operand= * NULL *

operator: SE
block number = 25
A←operand= stop
B←operand= * NULL *

operator: BNE
block number = 25
A←operand= v1t035
B←operand= * NULL *

operator: ENTER
block number = 26
A←operand= stop
B←operand= v1t036

operator: ADVANCE
block number = 27
A←operand= v1t037
B←operand= * NULL *

operator: SEIZE
block number = 28
A←operand= v1t038
B←operand= * NULL *

operator: ADVANCE
block number = 29
A←operand= gre1
B←operand= * NULL *

operator: RELEASE
block number = 30
A←operand= v1t039
B←operand= * NULL *

operator: ADVANCE
block number = 31
A←operand= v1t040
B←operand= * NULL *

operator: SEIZE
block number = 32
A←operand= v1t041
B←operand= * NULL *

operator: ADVANCE
block number = 33
A←operand= gre2
B←operand= * NULL *

operator: RELEASE
block number = 34
A←operand= v1t042
B←operand= * NULL *

operator: BR
block number = 35
A←operand= * NULL *
B←operand= loop

operator: GENERATE
block number = 36
A←operand= v1t043
B←operand= * NULL *

operator: TERMINATE
block number = 37
A←operand= v1t044
B←operand= * NULL *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for an unconditional TRANSFER
3               !
4                   simulate
5               1       storage         1
6               var1    variable        1
7               func1   function        r,d2
8               0.1,1/0.2,2
9               !
10      1               transfer        ,                       ! no label error
                                        *
                >> Required operands missing
11              ,       transfer                                ! syntax error
                    *
                >> Syntax error
12      2       label1  transfer        label2                  ! uncomditional TRANS has only B-operan
**d
                                            *
                >> Operand not valid in this version
13      3               transfer        ,label3                 ! ok
14      4       label3  transfer        ,1                      ! illegal label
                                        *
                >> Illegal specification of label
15      5       label4  transfer        A,label5                ! no A-operand allowed
                                        *
                >> Operand not valid in this version
16              1       transfer        ,label4                 ! illegal label
                    *
                >> Multiply defined label
17      6       label5  transfer        ,label6,c               ! no C-operand allowed
                                            *
                >> Syntax error
18      7       label6  transfer        ,,c                     ! no B-operand
                                        *
                >> Syntax error
19      8       label7  transfer        ,,c,d                   ! no B-operand
                                        *
                >> Syntax error
20      9       label8  transfer        ,v*var1                 ! illegal label
21                  end
```

```
**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: 1
        Label attribute: STORAGE
        Length of information: 1

Label: var1
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: label1
        Label attribute: LABEL

Label: label3
        Label attribute: LABEL

Label: label4
        Label attribute: LABEL

Label: label5
        Label attribute: LABEL

Label: label6
        Label attribute: LABEL

Label: label7
        Label attribute: LABEL

Label: label8
        Label attribute: LABEL
**************************************************
```

```
======================
START card information
======================
>>> Undefined label v*var1


======================
Pseudo Code Generated
======================


operator: BR
block number - 2
A←operand= * NULL *
B←operand= label3

operator: BR
block number - 8
A←operand= * NULL *
B←operand= v*var1

Total number of errors = 10
Simulation phase will not occur due to ERRORS
*       GPSS exiting     *
```

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       parsing test for VARIA8LE statement
3               !
4                   simulate
5               func1   function        r,d2
6               0.1,1/0.2,2
7               !
8               var1    variable        1*2+3-4/5               !ok
9               var2    variable        1+v*var1+3              !ok
10              var3    variable        v*var2+p*1+fn*func1     !ok
11              var4    variable        p*1-v*var3              !ok
12              var5    variable        v*var5*v*var2           !cannot use same variable
                                                *
                >> Undefined operand
13                      variable        1+2                     !required label missing
                                    *
                >> Label required
14              vara    variable        1+*2                    !arithmetic definition error
                                  *
                >> Syntax error
15              varb    variable        1+(2*3)                 !arithmetic definition error
                                   *
                >> Syntax error
                                  *
                >> Operand is not valid
16              varc    variable        +1*2                    !arithmetic definition error
                                  *
                >> Illegal variable definition
17              vard    variable        22+30                   !ok
18              vare    variable        2a+3                    !illegal operand
                                    *
                >> Syntax error
19              var6    variabl         3                       !syntax error
                                 *
                >> Syntax error
20              var7    variable        a+b                     !illegal operand
                                  *
                >> Syntax error
                                  *
                >> Required operands missing
21              var8    variable        s*2+1                   !illegal SNA
                                   *
                >> Syntax error
                                  *
                >> Required operands missing
22              var9    variable        1,2                     !arithmetic definition error
                                  *
                >> Syntax error
23              var10   variable        fn*func1+2              !undefined operand
24              var11   variable        1 1                     !syntax error
                                   *
                >> Operand is not valid
25              var12   variable        1-2                     !ok
26              var13   variable        1/2                     !ok
27              var14   variable        -1                      !ok
28              var15   variable        -fn*func1               !ok
29              var16   variable        -1/2*3                  !ok
30              var17   variable        2/3+4/5+5*6*3           !ok
31              var18   variable        2/3+-4                  !ok, (2/3) + -(4)
32              var19   variable        2/v*var6+-v*var7        !ok
33              var20   variable        1--v*var13              !ok, 1- ( -(var13))
34              var21   variable        v*var13*var14           !illegal specification of variable
                                                *
                >> Undefined operand
35              var22   variable        1/-2                    !ok
36              var23   variable        1/*23                   !arithmetic definition error
                                   *
                >> Illegal variable definition
37              var24   variable        34*-/34                 !arithmetic definiton error
                                    *
                >> Illegal variable definition
38              var25   variable        8*v*var5/v*var22        !ok
39              var26   variable        -v*var14                !ok
```

```
40              var27   variable        var14 + var15           !illegal spec of variable
                                *
                >> Syntax error
                                *
                >> Required operands missing
41              var27a  variable        zebra + wolf            !illegal spec of whatever
                                *
                >> Syntax error
                                *
                >> Required operands missing
42              var28   variable        1 / 2                   !ok
43              var29   variable        1+2+3+4+5+6+            !not ok, hanging arithmetic operator
                                                *
                >> Operand is not valid
44              var30   variable        65*59/45/64             !ok, ((65*59)/45)/64
45              end
```

```
**************************************************

Symbol Table Contents after First Pass Completion
==================================================

Label: func1
        Label attribute: FUNCTION
        Number of points in the function: 2
                0.100000        1
                0.200000        2
        Length of information: 2

Label: var1
        Label attribute: VARIABLE
        Length of information: 20
        Information stored:
                v1t000:3:+:v1t001:-:

Label: v1t000
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                1:2:*:

Label: v1t001
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                4:5:/:

Label: var2
        Label attribute: VARIABLE
        Length of information: 15
        Information stored:
                1:v*var1:+:3:+:

Label: var3
        Label attribute: VARIABLE
        Length of information: 24
        Information stored:
                v*var2:p*1:+:fn*func1:+:

Label: v1t002
        Label attribute: PARAMETER
        Information stored:
                1

Label: var4
        Label attribute: VARIABLE
        Length of information: 13
        Information stored:
                p*1:v*var3:-:

Label: v1t003
        Label attribute: PARAMETER
        Information stored:
                1

Label: var5
        Label attribute: VARIABLE

Label: vara
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: varb
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: varc
        Label attribute: VARIABLE
```

```
Label: vard
        Label attribute: VARIABLE
        Length of information: 8
        Information stored:
                22:30:+:

Label: vare
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                2:

Label: var6
        Label attribute: LABEL

Label: var7
        Label attribute: VARIABLE

Label: var8
        Label attribute: VARIABLE

Label: var9
        Label attribute: VARIABLE
        Length of information: 2
        Information stored:
                1:

Label: var10
        Label attribute: VARIABLE
        Length of information: 13
        Information stored:
                fn*func1:2:+:

Label: var11
        Label attribute: VARIABLE

Label: var12
        Label attribute: VARIABLE
        Length of information: 6
        Information stored:
                1:2:-:

Label: var13
        Label attribute: VARIABLE
        Length of information: 7
        Information stored:
                v1t004:

Label: v1t004
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                1:2:/:

Label: var14
        Label attribute: VARIABLE
        Length of information: 4
        Information stored:
                1:~:

Label: var15
        Label attribute: VARIABLE
        Length of information: 11
        Information stored:
                fn*func1:~:

Label: var16
        Label attribute: VARIABLE
        Length of information: 7
        Information stored:
                v1t006:

Label: v1t005
        Label attribute: TEMPORARY
        Length of information: 8
```

```
        Information stored:
                1:~:2:/:

Label: v1t006
        Label attribute: TEMPORARY
        Length of information: 11
        Information stored:
                v1t005:3:*:

Label: var17
        Label attribute: VARIABLE
        Length of information: 25
        Information stored:
                v1t007:v1t008:+:v1t010:+:

Label: v1t007
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                2:3:/:

Label: v1t008
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                4:5:/:

Label: v1t009
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                5:6:*:

Label: v1t010
        Label attribute: TEMPORARY
        Length of information: 11
        Information stored:
                v1t009:3:*:

Label: var18
        Label attribute: VARIABLE
        Length of information: 13
        Information stored:
                v1t011:4:~:+:

Label: v1t011
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                2:3:/:

Label: var19
        Label attribute: VARIABLE
        Length of information: 18
        Information stored:
                v1t012:v*var7:~:+:

Label: v1t012
        Label attribute: TEMPORARY
        Length of information: 11
        Information stored:
                2:v*var6:/:

Label: var20
        Label attribute: VARIABLE
        Length of information: 14
        Information stored:
                1:v*var13:~:-:

Label: var21
        Label attribute: VARIABLE

Label: var22
        Label attribute: VARIABLE
        Length of information: 7
        Information stored:
```

```
                v1t013:

Label: v1t013
        Label attribute: TEMPORARY
        Length of information: 8
        Information stored:
                1:2:~:/:

Label: var23
        Label attribute: VARIABLE

Label: v1t014
        Label attribute: TEMPORARY
        Length of information: 2
        Information stored:
                1:

Label: var24
        Label attribute: VARIABLE

Label: v1t015
        Label attribute: TEMPORARY
        Length of information: 3
        Information stored:
                34:

Label: var25
        Label attribute: VARIABLE
        Length of information: 7
        Information stored:
                v1t017:

Label: v1t016
        Label attribute: TEMPORARY
        Length of information: 11
        Information stored:
                B:v*var5:*:

Label: v1t017
        Label attribute: TEMPORARY
        Length of information: 17
        Information stored:
                v1t016:v*var22:/:

Label: var26
        Label attribute: VARIABLE
        Length of information: 10
        Information stored:
                v*var14:~:

Label: var27
        Label attribute: VARIABLE

Label: var27a
        Label attribute: VARIABLE

Label: var28
        Label attribute: VARIABLE
        Length of information: 7
        Information stored:
                v1t018:

Label: v1t018
        Label attribute: TEMPORARY
        Length of information: 6
        Information stored:
                1:2:/:

Label: var29
        Label attribute: VARIABLE
        Length of information: 22
        Information stored:
                1:2:+:3:+:4:+:5:+:6:+:

Label: var30
        Label attribute: VARIABLE
```

```
        Length of information: 7
        Information stored:
                v1t021:

Label: v1t019
        Label attribute: TEMPORARY
        Length of information: 8
        Information stored:
                65:59:*:

Label: v1t020
        Label attribute: TEMPORARY
        Length of information: 12
        Information stored:
                v1t019:45:/:

Label: v1t021
        Label attribute: TEMPORARY
        Length of information: 12
        Information stored:
                v1t020:64:/:
************************************************
```

```
======================
START card information
======================


======================
Pseudo Code Generated
======================


Total number of errors = 22
Simulation phase will not occur due to ERRORS
*        GPSS exiting     *
```

7.3.3. Simulator and Statistical Results.

Test Case 1: *No discrepancies.*

```
expon function    r,d3
.01,1/.05,2/.09,3
    generate   fn*expon
    terminate  1

    start  1
```

R.I.T. GPSS compiler

absolute clock = 3

block	current	total
1	0	1
2	0	1

Thesis GPSS compiler

absolute clock = 3

block	current	total
1	0	1
2	0	1

Test Case 2: *No discrepancies.*

```
expon function    r,d3
.01,1/.05,2/.09,3
    generate   fn*expon
    terminate  1

    start  20
```

R.I.T. GPSS compiler

absolute clock = 60

block	current	total
1	0	20
2	0	20

Thesis GPSS compiler

absolute clock = 60

block	current	total
1	0	20
2	0	20

Test Case 3: *Absolute clock discrepancy attributed to random number generation.*

```
expon function    r,d3
.01,1/.05,2/.09,3
    generate   fn*expon
    terminate  1

    start  100
```

R.I.T. GPSS compiler

absolute clock = 300

block	current	total
1	0	100
2	0	100

Thesis GPSS compiler

absolute clock = 296

block	current	total
1	0	100
2	0	100

Test Case 4: *No discrepancy.*

```
expon function    r,d3
.01,1/.05,2/.09,3
    generate   fn*expon,2
    terminate  1

    start  1
```

R.I.T. GPSS compiler

absolute clock = 1

block	current	total
1	0	1
2	0	1

Thesis GPSS compiler

absolute clock = 1

block	current	total
1	0	1
2	0	1

Test Case 5: *Absolute clock discrepancy attributed to random number generation.*

```
expon function    r,d3
.01,1/.05,2/.09,3
    generate   fn*expon,2
    terminate  1

    start  20
```

R.I.T. GPSS compiler

absolute clock = 53

block	current	total
1	0	20
2	0	20

Thesis GPSS compiler

absolute clock = 56

block	current	total
1	0	20
2	0	20

Test Case 6: *Absolute clock discrepancy attributed to random number generation.*

```
expon function    r,d3
.01,1/.05,2/.09,3
    generate   fn*expon,2
    terminate  1

    start  100
```

R.I.T. GPSS compiler

absolute clock = 315

block	current	total
1	0	100
2	0	100

Thesis GPSS compiler

absolute clock = 296

block	current	total
1	0	100
2	0	100

Test Case 7: *Absolute clock discrepancy attributed to random number generation.*

```
expon  function    r,d3
.01,1/.05,2/.09,3
plus   function    r,d4
.25,1/.50,2/.75,3/.9999,4
    generate   fn*expon,fn*plus
    terminate  1

    start  1
```

R.I.T. GPSS compiler

absolute clock = 12

block	current	total
1	0	1
2	0	1

Thesis GPSS compiler

absolute clock = 3

block	current	total
1	0	1
2	0	1

Test Case 8: *There is a large discrepancy seen in the absolute clock time. My conclusion is the R.I.T. GPSS simulator is in error when it evaluates the functions. The functions have been created so the function* ***expon*** *has a value of 3 90% of the time and the function* ***plus*** *has a value which is evenly distributed in the range 1 to 4. Thus:*

expon *= 3 avg.*
plus *= 2.5 avg.*
*generate = 3*2.5 = 7.5*
*generate * time units = 7.5*20 =* ***150*** **avg.**

```
expon  function    r,d3
.01,1/.05,2/.09,3
plus   function    r,d4
.25,1/.50,2/.75,3/.9999,4
    generate   fn*expon,fn*plus
    terminate  1

    start  20
```

R.I.T. GPSS compiler

absolute clock = 240

block	current	total
1	0	20
2	0	20

Thesis GPSS compiler

absolute clock = 150

block	current	total
1	0	20
2	0	20

Test Case 9: *There is a large discrepancy seen in the absolute clock time. My conclusion is the R.I.T. GPSS simulator is in error when it evaluates the functions. The functions have been created so the function* ***expon*** *has a value of 3 90% of the time and the function* ***plus*** *has a value which is evenly distributed in the range 1 to 4. Thus:*

expon *= 3 avg.*
plus *= 2.5 avg.*
*generate = 3*2.5 = 7.5*
*generate * time units = 7.5*100 =* **750 avg.**

```
expon function    r,d3
.01,1/.05,2/.09,3
plus   function    r,d4
.25,1/.50,2/.75,3/.9999,4
    generate   fn*expon,fn*plus
    terminate  1

    start  100
```

R.I.T. GPSS compiler

absolute clock = 1200

block	current	total
1	0	100
2	0	100

Thesis GPSS compiler

absolute clock = 726

block	current	total
1	0	100
2	0	100

Test Case 10: *The discrepancy seen in both the absolute clock time and the block totals are attributed to the difference in evaluation of the* ***plus*** *function. Please refer to Test Case's 8 and 9 for an in-depth explanation.*

```
expon function    r,d3
.01,1/.05,2/.09,3
plus   function    r,d4
.25,1/.50,2/.75,3/.9999,4
    generate   fn*expon
    terminate  1

    generate   fn*plus
    terminate

    start  100
```

R.I.T. GPSS compiler

absolute clock = 172

block	current	total
1	1	58
2	0	57
3	0	43
4	0	43

Thesis GPSS compiler

absolute clock = 132

block	current	total
1	0	44
2	0	44
3	0	56
4	0	56

Test Case 11: *No discrepancies.*

```
var1    variable    100/50*3+2
    generate    v*var1
    terminate   1

    start   1
```

R.I.T. GPSS compiler

absolute clock = 8

block	current	total
1	0	1
2	0	1

Thesis GPSS compiler

absolute clock = 8

block	current	total
1	0	1
2	0	1

Test Case 12: *No discrepancies.*

```
var1    variable    100/50*3+2
    generate    v*var1
    terminate   1

    start   20
```

R.I.T. GPSS compiler

absolute clock = 160

block	current	total
1	0	20
2	0	20

Thesis GPSS compiler

absolute clock = 160

block	current	total
1	0	20
2	0	20

Test Case 13: *No discrepancies.*

```
var1    variable    100/50*3+2
    generate    v*var1
    terminate   1

    start   100
```

R.I.T. GPSS compiler

absolute clock = 800

block	current	total
1	0	100
2	0	100

Thesis GPSS compiler

absolute clock = 800

block	current	total
1	0	100
2	0	100

Test Case 14: *There is a large discrepancy in both the absolute clock time and the total contents of the blocks. An analysis of the generate blocks shows two transactions being created every five time units. Therefore, the run termination count will be reached after 250 times units + advance block time. Again, the conclusion is the R.I.T. GPSS simulator is in error.*

```
generate   5
assign     1,+5
advance    p*1
terminate  1

generate   5
assign     1,+2
advance    p*1
terminate  1

start  100
```

R.I.T. GPSS compiler

absolute clock = 47

block	current	total
1	1	10
2	0	9
3	1	9
4	0	8
5	1	10
6	0	9
7	0	9
8	0	9

Thesis GPSS compiler

absolute clock = 255

block	current	total
1	1	51
2	0	50
3	0	50
4	0	50
5	1	51
6	0	50
7	0	50
8	0	50

Test Case 15: *The current block 1 discrepancy is due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time.*

```
generate   5
assign     1,·2
assign     1,+5
advance    p*1
terminate  1

start  100
```

R.I.T. GPSS compiler

absolute clock = 503

block	current	total
1	1	101
2	0	100
3	0	100
4	0	100
5	0	100

Thesis GPSS compiler

absolute clock = 503

block	current	total
1	0	100
2	0	100
3	0	100
4	0	100
5	0	100

Test Case 16: *The current block 1 discrepancy is due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time.*

```
var1    variable    2
var2    variable    2+3
    generate    v*var1
    advance     v*var2
    terminate   1

    start   100
```

R.I.T. GPSS compiler

absolute clock = 205

block	current	total
1	1	103
2	2	102
3	0	100

Thesis GPSS compiler

absolute clock = 205

block	current	total
1	0	102
2	2	102
3	0	100

Test Case 17: *The block discrepancy is due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time.* All other discrepancies in block counts are due to the philosophy difference on when to exit the model.

```
        generate    5
label1  assign      1,+5
        advance     p*1
        transfer    ,label1

        generate    100
        terminate   1

        start   1
```

R.I.T. GPSS compiler

absolute clock = 100

block	current	total
1	1	20
2	0	79
3	19	79
4	0	60
5	0	1
6	0	1

Thesis GPSS compiler

absolute clock = 100

block	current	total
1	0	20
2	0	85
3	20	85
4	0	65
5	0	1
6	0	1

Test Case 18: *Both the block and the storage statistic discrepancies are due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time.*

```
var2    variable    2+3
1       storage     100
    generate    2
    assign      1,+1
    enter       1,v*var2
    advance     3
    leave       1,5
    terminate   1

    start   100
```

R.I.T. GPSS compiler

absolute clock = 203

block	current	total
1	1	102
2	0	101
3	0	101
4	1	101
5	0	100
6	0	100

Thesis GPSS compiler

absolute clock = 203

block	current	total
1	0	101
2	0	101
3	0	101
4	1	101
5	0	100
6	0	100

	Storage	Capacity	Average Contents	Average Util	Total Entries	Average T/Trans	Maximum Contents	Current Contents
R.I.T. :	1	100	7.414	0.074	505	2.98	10	5
Thesis :	1	100	7.389	0.074	505	2.97	10	5

Test Case 19: *Both the block and the queue statistic discrepancies are due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time.*

```
var2    variable    2+3
    generate    2
    assign      1,+1
    queue       1,v*var2
    advance     5
    depart      1,v*var2
    terminate   1

    start   100
```

R.I.T. GPSS compiler

absolute clock = 205

block	current	total
1	1	103
2	0	102
3	0	102
4	2	102
5	0	100
6	0	100

	Queue	Maximum Contents	Average Contents	Total Entries	Zero Entries	Average T/Trans	Current Contents
R.I.T. :	1	15	12.293	510	0	4.941	10
Thesis :	1	15	12.195	510	0	4.902	10

Thesis GPSS compiler

absolute clock = 205

block	current	total
1	0	102
2	0	102
3	0	102
4	2	102
5	0	100
6	0	100

Test Case 20: *Both the block and the facility statistic discrepancies are due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time and when is the correct time to exit the model.*

```
generate    2
assign      1,+1
seize       1
advance     20
release     1
terminate   1

start 20
```

R.I.T. GPSS compiler

absolute clock = 402

block	current	total
1	1	201
2	180	200
3	0	20
4	0	20
5	0	20
6	0	20

Thesis GPSS compiler

absolute clock = 440

block	current	total
1	0	220
2	200	220
3	0	20
4	0	20
5	0	20
6	0	20

	Facility	Average Utilization	Number Entries	Average T/Trans	Seizing Contents
R.I.T. :	1	.995	20	20.000	.
Thesis :	1	.910	20	20.000	3

Test Case 21: *The block discrepancy is due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time.*

```
        generate    2
label   assign      1,+1
        test g      p*1,21,label
        terminate

        start       20,,10
```

R.I.T. GPSS compiler

absolute clock = 20

block	current	total
1	1	11
2	0	220
3	0	220
4	2	10

absolute clock = 40

block	current	total
1	1	21
2	0	440
3	0	440
4	2	20

Thesis GPSS compiler

absolute clock = 22

block	current	total
1	1	11
2	0	220
3	0	220
4	2	10

absolute clock = 40

block	current	total
1	0	20
2	0	440
3	0	440
4	2	20

Test Case 22: *Both the block and the facility statistic discrepancies are due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time and when is the correct time to exit the model.*

```
        generate    2
label   assign      1,+1
        advance     4
        seize       1
        gate u      1,label
        advance     10
        release     1
        terminate

        generate    4
        terminate   1

    start   20
```

R.I.T. GPSS compiler

absolute clock = 80

block	current	total
1	1	40
2	0	39
3	31	39
4	0	8
5	0	8
6	1	8
7	0	7
8	0	7
9	0	20
10	0	20

Thesis GPSS compiler

absolute clock = 80

block	current	total
1	0	40
2	0	40
3	33	40
4	0	7
5	0	7
6	1	7
7	0	6
8	0	6
9	0	20
10	0	20

	Facility	Average Utilization	Number Entries	Average T/Trans	Seizing Contents
R.I.T. :	1	.925	8	9.250	9
Thesis :	1	.750	7	8.571	4

Test Case 23: *Both the block and the facility statistic discrepancies are due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time and when is the correct time to exit the model.*

```
        generate    2
label   assign      1,+1
        advance     4
        gate nu     1,label
        seize       1
        advance     10
        release     1
        terminate   1

    start   20
```

R.I.T. GPSS compiler

absolute clock = 206

block	current	total
1	1	103
2	0	2122
3	82	2122
4	0	2040
5	0	20
6	0	20
7	0	20
8	0	20

	Facility	Average Utilization	Number Entries	Average T/Trans	Seizing Contents
R.I.T. :	1	.971	20	10.000	.
Thesis :	1	.820	20	10.000	5

Thesis GPSS compiler

absolute clock = 244

block	current	total
1	0	122
2	0	2122
3	62	2122
4	0	2060
5	0	20
6	0	20
7	0	20
8	0	20

Test Case 24: *Both the block and the storage statistic discrepancies are due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time and when is the correct time to exit the model.*

```
1       storage     100

        generate    2
label   assign      1,+1
        advance     4
        gate se     s*1,label
        enter       s*1
        advance     10
        leave       s*1
        terminate   1

    start   20
```

R.I.T. GPSS compiler

absolute clock = 206

block	current	total
1	1	103
2	0	2122
3	82	2122
4	0	2040
5	0	20
6	0	20
7	0	20
8	0	20

Thesis GPSS compiler

absolute clock = 244

block	current	total
1	0	122
2	0	2122
3	62	2122
4	0	2060
5	0	20
6	0	20
7	0	20
8	0	20

	Storage	Capacity	Average Contents	Average Util	Total Entries	Average T/Trans	Maximum Contents	Current Contents
R.I.T. :	1	100	.971	0.010	20	10.000	1	0
Thesis :	1	100	.820	0.008	20	10.000	1	0

Test Case 25: *Both the block and the storage statistic discrepancies are due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time.*

```
1       storage     100

        generate    2
label   assign      1,+1
        advance     4
        enter       s*1
        gate sf     s*1,label
        advance     10
        leave       s*1
        terminate   1

    start   20
```

R.I.T. GPSS compiler

absolute clock = 242

block	current	total
1	1	121
2	0	219
3	100	219
4	0	119
5	0	119
6	0	20
7	0	20
8	0	20

Thesis GPSS compiler

absolute clock = 288

block	current	total
1	0	144
2	0	243
3	124	243
4	0	119
5	0	119
6	0	20
7	0	20
8	0	20

	Storage	Capacity	Average Contents	Average Util	Total Entries	Average T/Trans	Maximum Contents	Current Contents
R.I.T. :	1	100	87.727	0.877	119	178.403	100	99
Thesis :	1	100	86.923	0.825	119	174.429	100	99

Test Case 26: *Both the block and the storage statistic discrepancies are due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time and when is the correct time to exit the model.*

```
1       storage     100

        generate    2
label   assign      1,+1
        advance     4
        gate snf    s*1,label
        enter       s*1,100
        advance     10
        leave       s*1,100
        terminate   1

    start   20
```

R.I.T. GPSS compiler

absolute clock = 206

block	current	total
1	1	103
2	0	2122
3	82	2122
4	0	2040
5	0	20
6	0	20
7	0	20
8	0	20

Thesis GPSS compiler

absolute clock = 244

block	current	total
1	0	122
2	0	2122
3	62	2122
4	0	2060
5	0	20
6	0	20
7	0	20
8	0	20

	Storage	Capacity	Average Contents	Average Util	Total Entries	Average T/Trans	Maximum Contents	Current Contents
R.I.T. :	1	100	97.087	.971	2000	10.000	100	0
Thesis :	1	100	81.967	.820	2000	10.000	100	0

Test Case 27: *Both the block and the storage statistic discrepancies are due to the different philosophies between the simulators of adding the transaction to the current events chain before checking the absolute clock time and when is the correct time to exit the model.*

```
1       storage     100

        generate    2
label   assign      1,+1
        advance     4
        enter       s*1
        gate sne    s*1,label
        advance     10
        leave       s*1
        terminate   1

    start   20
```

R.I.T. GPSS compiler

absolute clock = 54

block	current	total
1	1	27
2	0	26
3	2	26
4	0	24
5	0	24
6	4	24
7	0	20
8	0	20

	Storage	Capacity	Average Contents	Average Util	Total Entries	Average T/Trans	Maximum Contents	Current Contents
R.I.T. :	1	100	4.074	.041	24	9.167	5	4
Thesis :	1	100	3.703	.037	25	8.000	6	5

Thesis GPSS compiler

absolute clock = 54

block	current	total
1	0	27
2	0	27
3	2	27
4	0	25
5	0	25
6	5	25
7	0	20
8	0	20

Final Test Case:

```
*       GPSS V1.0       - May,1984
*     author: J. Van Dellon
1               !
2               !       Purpose:
3               !               The following program is used for a generic debugging tool
4               !       for the GPSS compiler/simulator thesis.  The following program
5               !       depicts a traffic problem over a bridge where one lane has been shut
6               !       off and the traffic is redirected through the other lane by means of
7               !       a traffic light at each end of the bridge.
8               !
9               !       Written:
10              !               Jacky Van Dellon        March 15, 1984
11              !
12              !-------------------------------------------------------------------------
13              !
14              !       Parameters:
15              !               4       - keep count of cars in direction 2
16              !               5       - keep count of cars in direction 1
17              !
18                  simulate
19              !
20              !       Traffic lights problem
21              !       ======================
22              expon   function        r,d23
23              0.0,0/0.1,10/0.2,22/0.3,35/0.4,50/0.5,69/0.6,91/0.7,120/0.8,160/0.84,183/
24              0.88,212/0.9,230/0.92,252/0.94,281/0.95,299/0.96,320/0.97,350/0.98,390/
25              0.99,460/0.995,530/0.998,620/0.999,700/0.9997,800
27              gre1    variable        600     ! length of time for green light in direction 1
28              gre2    variable        450     ! length of time for green light in direction 2
29              stop    storage         1       ! storage to restrict flow of transactions
30              1       storage         30      ! storage used for entering cars in direction 1
31              2       storage         30      ! storage used for entering cars in direction 2
32              !
33              !       traffic from direction 1
34              !       ========================
35      1               generate 90,fn*expon            !generate cars
36      2               queue   1                       !enter queue
37      3               seize   1                       !seize starting place
38      4               gate u  1                       !is light green
39      5               transfer        ,delay          !null transfer
40              !
41      6       null1   advance 100                     !null block
42              !
43      7       delay   advance 20                      !start car
44      8               assign  5,+1                    !keep count of cars
45      9               enter   1,1                     !enter storage
46      10              depart  1                       !leave queue
47      11              release 1                       !release starting place
48      12              leave   1                       !leave storage
49      13              terminate                       !car is over bridge
50              !
51              !       traffic from direction 2
52              !       ========================
53      14              generate 120,fn*expon           !generate cars
54      15              queue   2                       !enter queue
55      16              seize   2                       !seize starting position
56      17              gate u  2                       !is light 2 green
57      18              test g  p*5,10,around           !check for ten cars in the counter
58              !
59      19      null2   advance 100                     !null block
60              !
61      20      around  advance 20                      !start car
62      21              assign  4,+1                    !count cars in direction 2
63      22              depart  2                       !leave starting place
64      23              release 2                       !release starting place
65      24              terminate                       !car is over bridge
66              !
67              !       traffic lights
68              !       ==============
69      25              generate 1                      !just start this sequence up
70      26              gate se s*stop,term             !make sure only one goes through
71      27              enter   s*stop                  !one is through and starts cycle
72              !
73      28      loop    advance 550                     !both lights red
74      29              seize   1                       !light 1 becomes green
```

```
75      30              advance v*gre1                  !green time for 1
76      31              release 1                       !light one turns red
77      32              advance 550                     !both lights red
78      33              seize   2                       !light 2 becomes green
79      34              advance v*gre2                  !green time for 2
80      35              release 2                       !light two turns red
81      36              transfer ,loop                  !begin new light cycle
82      37      term    terminate
83              !
84              !       auxiliary clock
85              !       ===============
86      38              generate 36000                  !one transaction per hour
87      39              terminate 1                     !decrement
88              !
89              !       start
90              !       =====
91                  start               1
92              !
93                  end
```

Absolute clock: 36000

BLOCK COUNTS

Block	Current	Total	Block	Current	Total	Block	Current	Total
1	0	5	11	0	5	21	0	2
2	0	5	12	0	5	22	0	2
3	0	5	13	0	5	23	0	2
4	0	5	14	0	2	24	0	2
5	0	5	15	0	2	25	0	36000
6	0	0	16	0	2	26	1	36000
7	0	5	17	0	2	27	0	1
8	0	5	18	0	2	28	0	17
9	0	5	19	0	0	29	0	17
10	0	5	20	0	2	30	0	17

BLOCK COUNTS

Block	Current	Total	Block	Current	Total	Block	Current	Total
31	0	17						
32	1	17						
33	0	16						
34	0	16						
35	0	16						
36	0	16						
37	0	35998						
38	0	1						
39	0	1						

Facility	Average Utilization	Number Entries	Average Time/Tran	Seizing Trans.No.
1	0.286111	22	468.18182	3
2	0.201111	18	402.22222	33

Queue	Maximum Contents	Average Contents	Total Entries	Zero Entries	Average Time/Trans	Current Contents
2	2	0.074472	2	0	1340.500000	0
1	1	0.000005	5	0	0.037830	0

Storage	Capacity	Average Contents	Average Utilization	Entries	Average Time/Trans	Current Contents	Maximum Contents
stop	1	0.000000	0.000000	1	0.000000	1	1
1	30	0.000000	0.000000	5	0.000000	0	1

absolute clock = 36000

block	current	total	block	current	total	block	current	total
1	1	1	11	0	0	21	0	0
2	0	0	12	0	0	22	0	0
3	0	0	13	0	0	23	0	0
4	0	0	14	1	1	24	0	0
5	0	0	15	0	0	25	1	36000
6	0	0	16	0	0	26	0	35999
7	0	0	17	0	0	27	0	1
8	0	0	18	0	0	28	0	17
9	0	0	19	0	0	29	0	17
10	0	0	20	0	0	30	0	17

block	current	total
31	0	17
32	1	17
33	0	16
34	0	16
35	0	16
36	0	16
37	0	35998
38	0	1
39	0	1

	Storage	Capacity	Average Contents	Average Util	Total Entries	Average T/Trans	Maximum Contents	Current Contents
R.I.T. :	1	30	0	0.000	0	0.000	0	0
	stop	1	1.000	1.000	1	35999.0	1	1

	Facility	Average Utilization	Number Entries	Average T/Trans	Seizing Contents
R.I.T. :	1	.283	17	600.000	-
	2	.200	16	450.000	-

NO QUEUE STATISTICS WERE GENERATED.

8. Conclusions.

8.1. Problems Encountered and Solved.

The GPSS programmer had no easy way to transfer control from one part of the program to the other, given the original thesis proposal GPSS subset. Therefore, the TRANSFER statement was added to the initial subset of 18 block statements.

After the design phase of the pseudo language was complete, the decision was made to generate the pseudo code in the first pass to increase the efficiency of the compiler. Instead of the first pass strictly being parsing and the second pass strictly being code generation, the design was changed to parse and create the skeleton of the pseudo code in the first pass. The second pass would complete the pseudo code structure. Integration of the generator was a simple addition due to the modular design of the parser.

8.2. Discrepancies and Shortcomings of the System.

The main shortcoming of the GPSS compiler/simulator is the entire set of the GPSS language, and its options are not implemented. This subset effects both the style of the GPSS programmer and the ability to accurately represent all models.

The VAX/VMS C random number utility , although a valid generator, lacks the capability of the separate generators implemented in other GPSS compilers. The statistical results would not vary significantly, with the multiple generators, but they would give you a greater period than a single random number utility.

Due to the subset of the language implemented for this thesis, only the Current Events Chain and the Future Events Chain are used. The two chains not implemented are the Interrupt Chain and the User Chain.

The statistical output ***cannot*** be formatted by the user thus constraining the user to the standard informational output.

The GPSS compiler also allows ***no interrupts*** during the simulation. A useful addition for the user would be an interactive debugging tool.

8.3. Lessons Learned.

The thesis was designed to enhance my knowledge of code generation and symbol table design. This particular goal was achieved but several added benefits were received through this project: 1. The ability to research a subject and put these abstract concepts into workable programs, and 2. The knowledge that hours spent in program design are ***not*** wasted. The time spent planning will save many hours of redesign on the fly, therefore making the program into a *kludge*.

8.3.1. Alternative Approaches for Improved System.

Although the symbol table's binary tree structure was sufficient for the subset of the GPSS language defined by this thesis, the improved performance of a hash algorithm would be a necessity for any future expansion of this compiler.

The pseudo code generator was implemented because of my desire to learn, design and implement a form of code generation. It would be an interesting alternative to implement the GPSS language as an interpreter.

8.3.2. Suggestions for Future Extensions.

Many different aspects and powers of the GPSS language were not implemented in this thesis. The expansion of the subset of statements and the addition of options to current statements would make this compiler a viable tool for everyday use.

An admirable addition to the statistical output would be the ability for the user to format the statistical results.

A required enhancement to the GPSS compiler, as the language grows more complicated, would be the implementation of an interactive debugger.

8.3.3. Related Thesis Topics for the Future.

Although many of my graduate classes touched on the subject of hash algorithms, the task of researching various methods and implementing a hash algorithm would not be a trivial task. The performance of the hash algorithm could even be measured against a binary tree structure, such as implemented in this thesis. Therefore, a conclusion could be drawn as to whether a hash implementation is justified for this specific case, the GPSS language.

A pseudo code implementation really is for optimization purposes. There is a realm of possibilities for research papers and an implementation of an optimizer. Also, optimization of code is an area which computer classes touch lightly, and a thesis in this area would be a good way to accomplish a stronger, well rounded background.

Along with optimization of code, an area seldom delved into is the art of debugging. An interactive debugger, although not a project within itself, could be an integral part of a research project.

9. Bibliography.

Aho, Alfred V. and Ullmann, Jeffrey D., *Principles of Compiler Design*, Addison-Wesley Publishing Company, 1977

Barret, William A. and Couch, John D., *Compiler Construction: Theory and Practice*, Science Research Associates, Inc., 1979

Bassett, Sam, *Computer Design*: "Multipass compilers produce tight code", page 44-47, 1984

Brown, P.J., *Writing Interactive Compilers and Interpreters*, John Wiley and Sons, 1979

Cocke, John and Schwartz, J.T., *Programming Languages and Their Compilers: Preliminary Notes.* Second Revised Version, Curant Institute of Mathematical Sciences, New York University, 1970

Goos, G., and J. Hartmanis editted by, *Lecture Notes in Computer Science: Compiler Construction - An Advanced Course,* Second Edition, Springer-Verlag Berlin- Heidelburg, 1976

Gordon, G., *"A General Purpose Systems Simulator"*, IBM Systems Journal, Volume 1, Number 1, 1962

Gordon, G. and Efron, R., *"A General Purpose Digital Simulator and Examples of Its Application: Part I. Description of the Simulator"*. IBM Systems Journal, Volume 3, Number 1, 1964

Gould, R.L., *"GPSS/360 - An Improved General Purpose Simulator"*, IBM Systems Journal, Volume 8, Number 1, 1969

Herscovitch, H., and Schneider, T., *"GPSS III An Expanded General Purpose Simulator"*. IBM Systems Journal, Volume 4, Number 3, 1965

Hetzel, William C., *Program Testing Methods*, Prentice Hall, Inc., 1972

Lewis, T.G., *Distribution Sampling for Computer Simulation*, Lexington Books, D.C. Heath Company, 1975

Martin, M. David, *General Purpose Simulation System for VAX/VMS Reference Manual*, Simulation Software Design and Development, Ontario, Canada, 1981

Pratt, Terrance W., *Programming Languages: Design and Implementation.* Prentice Hall, Inc., 1975

Programming in VAX-11 C, Digital Equipment Corporation, 1982, Page 140

XEROX General Purpose Discrete Simulator (GPDS), Reference Manual, 1972

10. Footnotes.

1. ***Compiler Construction: Theory and Practice***, page 466, William A. Barrett and John D. Couch, Science Research Associates, 1979.

2. ***A General Purpose Systems Simulator***, page 18, Geoffrey Gordon, IBM Systems Journal, September 1962.

3. ***A General Purpose Systems Simulator***, page 20, Geoffrey Gordon, IBM Systems Journal, September 1962.

4. ***Simulation with GPSS and GPSS V***, page 1, P.A. Bobillier, B.C. Kahan, A.R. Probst, Prentice-Hall, Inc., 1976.

5. ***A General Purpose Digital Simulator and Examples of its Application: Part I - Description of a simulator***, page 22-23, R.EFRON and G. Gordon, IBM Systems Journal, Volume 3, 1964

6. ***Writing Interactive Compilers and Interpreters***, page 66, P.J. Brown, John Wiley, and Sons, 1979

7. ***Principles of Compiler Design***, page 327-349, Alfred V. Aho & Jeffrey D. Ullman, Addison-Wesley Publishing Company, 1977

8. ***Simulation with GPSS and GPSS V***, page 2, P.A. Bobillier, B.C. Kahan, A.R. Probst, Prentice-Hall, Inc., 1976.

9. ***GPSS/360 - an improved general purpose simulator***, page 18, R.L. Gould, IBM Systems Journal, Volume 8, Number 1, 1969.

10. ***Simulation with GPSS and GPSS V***, page 2, P.A. Bobillier, B.C. Kahan, A.R. Probst, Prentice-Hall, Inc., 1976.

11. ***XEROX General Purpose Discrete Simulator (GPDS)***, page 36, XEROX Corporation, 1972.

11. Appendices.

11.1. Appendix A - BNF grammar of the GPSS language.

<GPS-program> ::= <simulate-statement> <GPS-instruction> <end-statement>

<GPS-instruction> ::= [<label>] <GPS-statement> | <comment> | <GPS-instruction>

<GPS-statement> ::= <advance-statement> | <assign-statement> | <depart-statement> | <enter-statement> | <function-statement> | <gate-statement> | <generate-statement> | <leave-statement> | <queue-statement> | <release-statement> | <seize-statement> | <start-statement> | <storage-statement> | <terminate-statement> | <test-statement> | <transfer-statement> | <variable-statement>

<advance-statement> ::= [<label>] {ADVANCE | advance} {<numeric> | <operand-1>} [<comment>]

<assign-statement> ::= [<label>] {ASSIGN | assign} {<numeric> | <operand-1>}, {<numeric> | <operand-1>} [<comment>]

<depart-statement> ::= [<label>] {DEPART | depart} {<numeric> | <operand-1>}, {<numeric> | <operand-1>} [<comment>]

<end-statement> ::= END | end

<enter-statement> ::= [<label>] {ENTER | enter} {<storage-label> | <numeric> | <operand-1>}, {<numeric> | <operand-1>} [<comment>]

<function-statement> ::= [<label>] {FUNCTION | function} <random-def>, <mnemonic-4> [<comment>] cr <func-numeric>

<gate-statement> ::= [<label>] {GATE | gate} <mnemonic-2> {<storage-label> | <operand-1>}, <label> [<comment>]

<generate-statement> ::= [<label>] {GENERATE | generate} {<numeric> | <operand-1>}, {<numeric> | <operand-1>} [<comment>]

<leave-statement> ::= [<label>] {LEAVE | leave} {<storage-label> | <numeric> | <operand-1>}, {<numeric> | <operand-1>} [<comment>]

<queue-statement> ::= [<label>] {QUEUE | queue} {<numeric> | <operand-1>}, {<numeric> | <operand-1>} [<comment>]

<release-statement> ::= [<label>] {RELEASE | release} {<numeric> | <operand-1>} [<comment>]

<seize-statement> ::= [<label>] {SEIZE | seize} {<numeric> | <operand-1>} [<comment>]

<simulate-statement> ::= SIMULATE | simulate

<start-statement> ::= {START | start} <numeric>, <mnemonic-1>, <numeric> [<comment>]

<storage-statement> ::= {storage-label | numeric} {STORAGE | storage} <numeric> [<comment>]

<terminate-statement> ::= [<label>] {TERMINATE | terminate} [<numeric>] [<comment>]

<test-statement> ::= [<label>] {TEST | test} <mnemonic-3> <operand-1>, <operand-1>, <label> [<comment>]

<transfer-statement> ::= [<label>] {TRANSFER | transfer} ,<label> [<comment>]

<variable-statement> ::= [<label>] {VARIABLE | variable} <variable-def>

<mnemonic-1> ::= {NP | np} | {NULL | null} | "blank"

<mnemonic-2> ::= {U | u} | {NU | nu} | {SF | sf} | {SE | se} | {SNF | snf} | {SNE | sne}

<mnemonic-3> ::= {E | e} | {L | l} | {G | g}

<mnemonic-4} ::= D | d

<operand-1> ::= { {V | v} | {F | f} } * <label> | {P | p} * <numeric>

<random-def> ::= R | r

<func-numeric> ::= <numeric>, <numeric> | / <func-numeric>

<variable-def> ::= {<numeric> | <operand-1>} | { + | - | * | / } <variable-def>

<label> ::= <alpha> <alphanum>

<storage-label> ::= <numeric>

<alpha> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p |
q | r | s | t | u | v | w | x | y | z | A | B | C | D | E |
F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |
U | V | W | X | Y | Z | <alpha>

<alphanum> ::= <alpha> | <numeric> | <alphanum>

<numeric> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

11.2. Appendix B - individual parser test programs for GPSS statements.

SIMULATE:

simulate1.gps

```
            simulate    a           !no operands allowed
            simulatte               !misspelling, take as a label
            simulate                !duplicate simulate
    label   simulate                !no label allowed
```

simulate2.gps

```
            simulate    ,b          !syntax error due to comma
    #$*     simulate                !illegal label
    ,       simulate                !syntax error
```

simulate3.gps

```
            simulate                !ok
            end
```

END:

end1.gps

```
            simulate
            end     a           !no operands allowed
            endd                !misspelling, take as label
            end                 !duplicate end
    label   end                 !no label allowed
```

end2.gps

```
            simulate
            end     ,b          !syntax error due to comma
    #$*     end                 !illegal label
    ,       end                 !syntax error
```

end3.gps

```
            simulate                !ok
            end
```

START:

start.gps

```
          simulate
label     start   1                 !label not allowed
,         start                     !syntax error
          start   1                 !ok
          start   a                 !illegal operand
          start   ,NP,3             !required operand missing
          start   1, np             !ok
          start   1,,               !syntax correct, just no arguments
          start   1,,c              !illegal operand
          start   1,,2              !ok
          start   1,NULL            !ok
          start   1,bb              !illegal option
          start   1    np  1        !syntax error
          end
```

ADVANCE:

```
advance.gps
          simulate
    1     storage   1
    var1  variable  1
    func1 function  r,d2
    0.1,1/0.2,2
    !
    label advance                 !required operand missing
          advance   p1            !illegal operand
          advance   P1            !illegal operand
          advance   p*1           !ok, parameter
          advance   P*1           !ok, parameter
          advance   s*1           !not allowed SNA
          advance   S*1           !not allowed SNA
          advance   v*var1        !ok, variable
          advance   V*var1        !ok, variable
          advance   f*1           !invalid specification of function
          advance   fn*func1      !ok, function
          advance   FN*func1      !ok, function
          advance   1             !ok
          advance   a             !illegal operand
          advance   ,             !required operand missing
          advance   1,            !syntax error
          advance   400           !ok
    ,     advance                 !syntax error
     #$%  advance                 !illegal label
          advance   1   1         !syntax error
```

ASSIGN:

```
assign.gps
          simulate
1         storage    1
var1      variable   1
func1     function   r,d2
0.1,1/0.2,2
!
          assign     1,-1              !ok
          assign     p*1,+2            !ok
          assign     fn*func1,-1       !ok
          assign     v*var1,+2         !ok
          assign     1,+p*1            !ok
          assign     1,+fn*func1       !ok
          assign     1,-v*var1         !ok
          assign     ,-2               !required operand missing
          assign     1,+-3             !invalid specification of arithmetic
          assign     +1,2              !invalid A-operand
          assign     1+,2              !comma not in proper place
          assign     1,-               !no numeric specified
          assign     a,-1              !illegal A-operand
          assign     1,+b              !illegal B-operand
          assign     s*1,+1            !illegal SNA
          assign     1,-s*1            !illegal SNA
          assign     v*var2,+1         !undefined operand
          assign     1,+v*var2         !undefined operand
          end
```

DEPART:

depart.gps

```
          simulate
a         variable   1
b         variable   1
stor1     storage    20
func1     function   r, d2
0.1,1/0.2,2
!
          depart     s*stor1        !not a good queue definition
label     depart                    !required operand missing
          depart     1              !ok
          depart     a              !A-operand not valid
          depart     a,             !missing B-operand, but A missing
          depart     ,              !required operand missing
          depart     a, b           !illegal specification of SNA's
          depart     v*a,v*b        !ok, variable specifications
          depart     p1             !illegal operand
label     depart     1              !multiple labels
          depart     1,v*b          !ok
          depart     v*a,2          !ok
          depart     s*1            !not defined SNA
          depart     p*1,1          !ok, correct specification
          end
```

ENTER:

```
enter.gps
          simulate
    a     variable   1
    b     variable   1
    stor1 storage    20
    func1 function   r, d2
    0.1,1/0.2,2
    !
          enter      s*stor1        !ok, storage label specification
    label enter                     !required operand missing
          enter      1              !ok
          enter      a              !A-operand not valid
          enter      a,             !B-operand missing, but A missing
          enter      ,              !required operand missing
          enter      a, b           !illegal specification of SNA's's
          enter      v*a,v*b        !ok, variable specifications
          enter      p1             !illegal operand
    label enter      1              !multiple labels
          enter      1,v*b          !ok
          enter      v*a,2          !ok
          enter      s*1            !not defined SNA
          enter      p*1,1          !ok, correct specification
          end
```

GATE:

```
gate.gps
            simulate
    1       storage     1
    var1    variable    1
    func1   function    r,d2
    0.1,1/0.2,2
    !
    label1  gate u      s*1,label2      !invalid SNA
    label2  gate u      fn*func1,label3!ok
    label3  gate u      v*var1,label4   !ok
    label4  gate u      p*1,label1      !ok
    label5  gate u      1,label1        !ok
            gate u      1               !ok, shouldn't move until true
            gate nu     s*1,label2      !invalid SNA
            gate nu     fn*func1,label2!ok
            gate nu     v*var1,label2   !ok
            gate nu     p*1,label2      !ok
            gate nu     1,label1        !ok
            gate nu     1               !ok, shouldn't move until true
            gate sf     s*1,label1      !ok
            gate sf     p*1,label1      !illegal SNA
            gate sf     v*var1,label1   !illegal SNA
            gate sf     fn*func1,label1!illegal SNA
            gate sf     s*1             !ok, shouldn't move until true
            gate se     s*1,label1      !ok
            gate se     p*1,label1      !illegal SNA
            gate se     fn*func1,label1!illegal SNA
            gate se     v*var1,label1   !illegal SNA
            gate se     s*1             !ok, shouldn't move until true
            gate sf     1,1             !illegal SNA
            gate sf     s*1,1           !illegal label
            gate snf    s*1,label1      !ok
            gate snf    p*1,label1      !illegal SNA
            gate snf    v*var1,label1   !illegal SNA
            gate snf    fn*func1,label1!illegal SNA
            gate sne    s*1,label1      !ok
            gate sne    p*1,label1      !illegal SNA
            gate sne    v*var1,label1   !illegal SNA
            gate sne    fn*func1,label1!illegal SNA
            gate snf    1,1             !illegal SNA
            gate snf    s*1,1           !illegal label
            gate ru     1,label1        !illegal mnemonic
            end
```

GENERATE:

```
generate.gps
          simulate
1         storage    1
var1      variable   1
func1     function   r,d2
0.1,1/0.2,2
!
          generate   1              !ok
          generate   5,1            !ok
          generate   1,p*1          !can't cause no parameters are evaluated yet
          generate   1,fn*func1     !ok
          generate   1,v*var1       !ok
          generate   p*1,1          !can't cause no parameters are evaluated yet
          generate   v*var1,1       !ok
          generate   fn*func1,1     !ok
          generate   s*1,1          !illegal SNA
          generate   1,s*1          !illegal SNA
          generate   ,1             !required operand missing
          generate   1,             !syntax error
          generate   fn*func2       !undefined function
          generate   1   1          !syntax error
          generate   1,1,1          !syntax error
          end
```

LEAVE:

```
leave.gps
          simulate
    a     variable   1
    b     variable   1
    stor1 storage    20
    func1 function   r, d2
    0.1,1/0.2,2
    !
          leave      s*stor1       !ok, storage label specification
    label leave                    !required operand missing
          leave      1             !ok
          leave      a             !illegal A-operand
          leave      a,            !no B-operand, missing A-operand
          leave      ,             !required operand missing
          leave      a, b          !illegal specification of SNA's
          leave      v*a,v*b       !ok, variable specifications
          leave      p1            !illegal operand
    label leave      1             !multiple labels
          leave      1,v*b         !ok
          leave      v*a,2         !ok
          leave      s*1           !not defined SNA
          leave      p*1,1         !ok, correct specification
          end
```

QUEUE:

```
queue.gps
          simulate
    a     variable   1
    b     variable   1
    stor1 storage    20
    func1 function   r, d2
    0.1,1/0.2,2
    !
          queue      s*stor1        !ok, storage label specification
    label queue                     !required operand missing
          queue      1              !ok
          queue      a              !illegal A-operand
          queue      a,             !no B-operand, missing A-operand
          queue      ,              !required operand missing
          queue      a, b           !illegal specification of SNA's
          queue      v*a,v*b        !ok, variable specifications
          queue      p1             !illegal operand
    label queue      1              !multiple labels
          queue      1,v*b          !ok
          queue      v*a,2          !ok
          queue      s*1            !not defined SNA
          queue      p*1,1          !ok, correct specification
          end
```

RELEASE:

```
release.gps
          simulate
    1     storage   1
    jacky variable  1
    func1 function  r, d2
    0.1,1/0.2,2
    !
    label release                 !required operand missing
    1a    release   2             !illegal label
          release   1             !ok
          release   a             !not allowed SNA
          release   p1            !illegal specification of SNA
          release   1,            !syntax error
          release   ,1            !required operand missing
          release   P1            !illegal specification of SNA
          release   p*1           !ok, parameter
          release   P*1           !ok, parameter
          release   s*1           !not allowable SNA
          release   S*1           !not allowable SNA
          release   v*jacky       !ok, variable
          release   V*jacky       !ok, variable
          release   f*1           !facility not allowed
          release   FN*func1      !ok, function
          release   fn*func1      !ok, function
          release   p*1           !ok, parameter
          release   P*1           !ok, parameter
          release   1   1         !syntax error
    ,     release                 !syntax error
    @ # # release                 !illegal label
          end
```

SEIZE:

seize.gps

```
          simulate
1         storage    1
jacky     variable   1
func1     function   r, d2
0.1,1/0.2,2
!
label     seize                  !required operand missing
1a        seize      2           !illegal label
          seize      1           !ok
          seize      a           !not allowed SNA
          seize      p1          !illegal specification of SNA
          seize      1,          !syntax error
          seize      ,1          !required operand missing
          seize      P1          !illegal specification of SNA
          seize      p*1         !ok, parameter
          seize      P*1         !ok, parameter
          seize      s*1         !not allowable SNA
          seize      S*1         !not allowable SNA
          seize      v*jacky     !ok, variable
          seize      V*jacky     !ok, variable
          seize      f*1         !facility not allowed
          seize      FN*func1    !ok, function
          seize      fn*func1    !ok, function
          seize      p*1         !ok, parameter
          seize      P*1         !ok, parameter
          seize      1   1       !syntax error
,         seize                  !syntax error
@ # #     seize                  !illegal label
          end
```

TERMINATE:

```
terminate.gps
          simulate
    1     storage    1
    var1  variable   1
    func1 function   r,d2
    0.1,1/0.2,2
    !
    label  terminate                  !ok
           terminate                  !ok, default 1
           terminate  1               !ok
           terminate  ,               !syntax error
           terminate  a               !illegal operand
           terminate  2,              !syntax error
           terminate  3   4           !syntax error
           terminate  5               !ok
           terminateer                !misspelling, error
    ,      terminate                  !syntax error
    @ # #  terminate                  !illegal label
           terminate  P1              !illegal operand
           terminate  p1              !illegal operand
           terminate  p*1             !no SNA's allowed
           terminate  P*1             !no SNA's allowed
           terminate  s*1             !no SNA's allowed
           terminate  S*1             !no SNA's allowed
           terminate  v*1             !no SNA's allowed
           terminate  V*1             !no SNA's allowed
           terminate  fn*1      !no SNA's allowed
           terminate  FN*1      !no SNA's allowed
       end
```

TEST:

test.gps

```
          simulate
1         storage    1
var1      variable   1
func1     function   r,d2
0.1,1/0.2,2
!
label1    test e   p*1,p*2,label2          !ok
label2    test e   v*var1,p*1,label3       !ok
label3    test e   fn*func1,p*1,label1     !ok
          test e   1,p*1,label2            !ok
          test e   1,2,label3              !ok
          test e   1,p*1,label1            !ok
          test e   1,fn*func1,label2       !ok
          test e   1,v*var1,label3         !ok
          test l   1,2,label3              !ok
          test g   1,2,label3              !ok
          test ge  1,2,label3              !illegal option
          test l   s*1,1,label3            !illegal SNA
          test g   1,s*1,label3            !illegal SNA
          test e   ,1,label3               !required operand missing
          test l   1,,label3               !required operand missing
          test g   1,1                     !ok
          test e   1,1,label4              !incorrect label error second pass
          test l   a,1,label1              !illegal operand
          test g   1,a,label1              !illegal operand
          test e   v*var2,p*2,label2       !undefined operand
          end
```

TRANSFER:

```
transfer.gps
      simulate
   1       storage    1
   var1    variable   1
   func1   function   r,d2
   0.1,1/0.2,2
   !
           transfer   ,              !no label error
   ,       transfer                  !syntax error
   label1  transfer   label2         !unconditional transfer has only B-operand
           transfer   ,label3        !ok
   label3  transfer   ,1             !illegal label
   label4  transfer   A,label5       !no A-operand allowed
   1       transfer   ,label4        !illegal label
   label5  transfer   ,label6,c      !no C-operand allowed
   label6  transfer   ,,c            !no B-operand
   label7  transfer   ,,c,d          !no B-operand
   label8  transfer   ,v*var1        !illegal label
      end
```

STORAGE:

```
storage.gps
          simulate
          storage                    !missing  label
a         storage   2                !ok
1         storage   3                !ok
1a        storage   4                !illegal  label
1         storage   5                !duplicate  label
2         storage                    !required  operand  missing
3         storage   six              !illegal  operand
4         storage   7,               !syntax  error
5         storage   8,9              !syntax  error
6         storage   10  11           !syntax  error
,         storage                    !syntax  error
$%**      storage   1                !illegal  label
          end
```

VARIABLE:

```
variable.gps
          simulate
    func1 function   r,d2
    0.1,1/0.2,2
    !
    var1      variable   1*2+3-4/5                !ok
    var2      variable   1+v*var1+3               !ok
    var3      variable   v*var2+p*1+fn*func1      !ok
    var4      variable   p*1-v*var3               !ok
    var5      variable   v*var5*var*2             !cannot use same variable
              variable   1+2                      !required label missing
    vara      variable   1+*2                     !arithmetic definition error
    varb      variable   1+(2*3)                  !arithmetic definition error
    varc      variable   +1*2                     !arithmetic definition error
    vard      variable   22+30                    !ok
    vare      variable   2a+3                     !illegal operand
    var6      variabl    3                        !syntax error
    var7      variable   a+b                      !illegal operand
    var8      variable   s*2+1                    !illegal SNA
    var9      variable   1,2                      !arithmetic definition error
    var10     variable   fn*func2+2               !undefined operand
    var11     variable   1  1                     !syntax error
    var12     variable   1-2                      !ok
    var13     variable   1/2                      !ok
    var14     variable   -1                       !ok
    var15     variable   -fn*func1                !ok
    var16     variable   -1/2*3                   !ok
    var17     variable   2/3+4/5+5*6*3            !ok
    var18     variable   2/3+-4                   !ok, (2/3)+(-(4))
    var19     variable   2/v*var6+-v*var7         !ok
    var20     variable   1--v*var13               !ok, 1-(-(var13))
    var21     variable   v*var13*var14            !illegal specification of variable
    var22     variable   1/-2                     !ok
    var23     variable   1/*23                    !arithmetic definition error
    var24     variable   34*-/34                  !arithmetic definition error
    var25     variable   8*v*var5/v*var22         !ok
    var26     variable   -v*var14                 !ok
    var27     variable   var14 + var15            !illegal spec of variable
    var27a    variable   zebra + wolf             !illegal spec of whatever
    var28     variable   1 / 2                    !ok
    var29     variable   1+2+3+4+5+6+             !not ok, hanging arithmetic operator
    var30     variable   65*59/45/64              !ok, ((65*59)/45)/64
              end
```

11.3. Appendix C - Test program for GPSS parser

```
!
!   Purpose:
!       The following program is used for a generic debugging tool
!   for the GPSS compiler/simulator thesis.  The following program
!   depicts a traffic problem over a bridge where one lane has been shut
!   off and the traffic is redirected through the other lane by means of
!   a traffic light at each end of the bridge.
!
!   Written:
!       Jacky Van Dellon          March 15, 1984
!
!-------------------------------------------------------------------------------
!
!   Parameters:
!       4  -  keep count of cars in direction 2
!       5  -  keep count of cars in direction 1
!
    simulate
!
!   Traffic lights problem
!   = = = = = = = = = = = = =
expon function    r,d23
0.0,0/0.1,104/0.2,222/0.3,355/0.4,509/0.5,690/0.6,915/0.7,1200/0.8,1600/
0.84,1830/0.88,2120/0.9,2300/0.92,2520/0.94,2810/0.95,2990/0.96,3200/0.97,3500/
0.98,3900/0.99,4600/0.995,5300/0.998,6200/0.999,7000/0.9997,8000

gre1    variable    600             !length of time for green light in direction 1
gre2    variable    450             !length of time for green light in direction 2
stop    storage     1               !storage to restrict flow of transactions
1       storage     30              !storage used for entering cars in direction 1
2       storage     30              !storage used for entering cars in direction 2
!
!   traffic from direction 1
!   = = = = = = = = = = = = =
        generate    90,fn*expon     !generate cars
        queue       1               !enter queue
        seize       1               !seize starting place
        gate u      1               !is light green
        transfer    ,delay          !null transfer
!
null1   advance     100             !null block
!
delay   advance     20              !start car
        assign      5,+1            !keep count of cars
        enter       1,1             !enter storage
        depart      1               !leave queue
        release     1               !release starting place
        leave       1               !leave storage
        terminate                   !car is over the bridge
!
!   traffic from direction 2
!   = = = = = = = = = = = = =
        generate    120,fn*expon    !generate cars
        queue       2               !enter queue
```

```
        seize     2                 !seize starting place
        gate u    2                 !is light 2 green
        test g    p*5,10,around     !check for ten cars in the counter
!
null2   advance   100               !null block
!
around  advance   20                !start car
        assign    4,+1              !count cars in direction 2
        depart    2                 !leave starting place
        release   2                 !release starting place
        terminate                   !car is over the bridge
!
!   traffic lights
!   = = = = = = =
        generate  1                 !just start this sequence up
        gate se   s*stop,term       !make sure only one goes through
        enter     s*stop            !one is through and starts cycle
!
loop    advance   550               !both lights red
        seize     1                 !light 1 becomes green
        advance   v*gre1            !green time for 1
        release   1                 !light one turns red
        advance   550               !both lights red
        seize     2                 !light 2 becomes green
        advance   v*gre2            !green time for 2
        release   2                 !light two turns red
        transfer  ,loop             !begin new cycle
term    terminate
!
!   auxiliary clock
!   = = = = = = = = =
        generate  36000             !one transaction per hour
        terminate 1                 !decrement
!
!   start
!   = = =
    start   1
!
    end
```

11.4. Appendix D - Error table for the GPSS parser

>> **Required operand missing.**

An operand which is required for the block is not within the statement. Please refer to chapter 13, the User Manual, for the specific block operand requirements.

>> **Label not allowed.**

The block does not allow a label. Please refer to chapter 13, the User Manual, for the specific block requirements.

>> **Operand is not valid.**

The operand has not been specified in a syntactically correct manner. Please refer to chapter 13, the User Manual, for the specific operand specifications.

>> **Undefined operand.**

The user either has not specified the operand prior to use or there was an error in the initial specification of the operand. All operands must be specified before being used in the GPSS program.

>> **Syntax error.**

The block statement has a grammar error. Please refer to chapter 13, the User Manual, for the specific block statement syntax.

>> **Illegal option specified.**

The GATE or TRANSFER block statement has an illegal option specified. Please refer to Chapter 13 for a list of legal options.

>> **Illegal specification of label.**

Labels can only start with an alphabetic character, A through Z. The *only* exception to this rule is the label which defines a storage area. The STORAGE label can be a numeric character, 0 through 9.

>> Open error on input file.

The file specified on the command line cannot be found in the directory specified.

>> Multiply defined label.

The label specified on the block statement has been previously defined in the program.

>> Multiple SIMULATION statements.

Only one SIMULATION statement is required for a GPSS program.

>> Multiple END statements.

Only one END statement is required for a GPSS program.

>> 50 START statements allowed.

Only 50 start statements are allowed in this version of GPSS.

>> Label required.

A label is required for the block statement specified. Please refer to chapter 13 for more information on the block statement in question.

>> Illegal specification of numeric quantity.

An integer number has not been specified correctly. Check for alphabetic characters, A through Z, or punctuation marks in the number.

>> Illegal specification of operand.

An operand has not been specified correctly. Check for correct specification of an SNA, or delimiters specified incorrectly.

>> Label valid but no statement.

A label with no following block statement is illegal.

>> Premature end of function.

The function statement is not fully defined. Please refer to chapter 13 section 3 for a description of the function statement.

>> Illegal function definition.

This error can happen for a number of reasons: the floating point number in the data statement is not specified correctly, the integer number in the data statement is not specified correctly, or the syntax of the data statement was incorrect. Please refer to chapter 13 section 4 for more information on the FUNCTION statement.

>> Missing function definition.

The FUNCTION statement, *label* ***FUNCTION R, Dn***, has been specified but the data statement, giving the data points and function values, is not specified.

>> Operand not valid in this version.

The TRANSFER block implemented in this version of GPSS is the *unconditional* transfer. No A-operand is allowed.

>> Illegal variable definition.

Check the syntax of the variable definition. Remember no parentheses are allowed and the unary subtraction operator should be specified to the left of the associated number.

>> A maximum of 10 parameters are allowed.

The maximum number of transaction parameters are 10. Check the parameter number specification in the indicated block statement.

>> COMPILER error.

This error indicates a software error and should be considered a major error. Please notify the manager associated with your system.

>>> Undefined label {*ASCII label*}.

The second pass of the compiler cannot find the specified label.

>>> SIMULATOR ERROR <<< Storage undefined.

The simulation phase has encountered an undefined storage area. This mistake occurs when using some form of indirection when specifying a storage location.

11.5. Appendix E - Program to verify the random number uniform distribution

```
#include stdio

main()
    {
    int random;
    int i;
    int one, two, three, four, five, six, seven, eight, nine, ten;
    int eleven, twelve, thirteen, fourteen, fifteen, sixteen, seventeen;
    int eighteen, nineteen, twenty;
    float temp;
    float flrand;
    int rand();

    one=two=three=four=five=six=seven=eight=nine=ten=eleven=twelve=0;
    thirteen=fourteen=fifteen=sixteen=seventeen=eighteen=nineteen=twenty=0;

    for(i= 1 ; i <= 100000 ; i++)
        {
        if((i%5) == 0)
                {
        random= rand();
        temp= random;
        flrand= temp /  2147483647.;
        if((flrand > 0.0) && (flrand < .05))
                one++;
        else if((flrand >= 0.05) && (flrand < .10))
                two++;
        else if((flrand >= 0.10) && (flrand < .15))
                three++;
        else if((flrand > 0.15) && (flrand < .20))
                four++;
        else if((flrand > 0.20) && (flrand < .25))
                five++;
        else if((flrand > 0.25) && (flrand < .30))
                six++;
        else if((flrand > 0.30) && (flrand < .35))
                seven++;
        else if((flrand > 0.35) && (flrand < .40))
                eight++;
        else if((flrand > 0.40) && (flrand < .45))
                nine++;
        else if((flrand > 0.45) && (flrand < .50))
                ten++;
        else if((flrand > 0.50) && (flrand < .55))
                eleven++;
        else if((flrand > 0.55) && (flrand < .60))
                twelve++;
        else if((flrand > 0.60) && (flrand < .65))
                thirteen++;
        else if((flrand > 0.65) && (flrand < .70))
                fourteen++;
        else if((flrand > 0.70) && (flrand < .75))
                fifteen++;
        else if((flrand > 0.75) && (flrand < .80))
                sixteen++;
        else if((flrand > 0.80) && (flrand < .85))
                seventeen++;
        else if((flrand > 0.85) && (flrand < .90))
                eighteen++;
        else if((flrand > 0.90) && (flrand < .95))
                nineteen++;
        else if((flrand > 0.95) && (flrand < 1.00))
                twenty++;
        }
        }

    printf("Distribution of random numbers\n");
    printf("==============================\n");
    printf("\n");
    printf(" 0.00 - .04999                  %f\n\n",(one/20000.));
    printf(" 0.05 - .09999                  %f\n\n",(two/20000.));
    printf(" 0.10 - .14999                  %f\n\n",(three/20000.));
    printf(" 0.15 - .19999                  %f\n\n",(four/20000.));
    printf(" 0.20 - .24999                  %f\n\n",(five/20000.));
    printf(" 0.25 - .29999                  %f\n\n",(six/20000.));
    printf(" 0.30 - .34999                  %f\n\n",(seven/20000.));
```

```
printf(" 0.35 - .39999          %f\n\n",(eight/20000.));
printf(" 0.40 - .44999          %f\n\n",(nine/20000.));
printf(" 0.45 - .49999          %f\n\n",(ten/20000.));
printf(" 0.50 - .54999          %f\n\n",(eleven/20000.));
printf(" 0.55 - .59999          %f\n\n",(twelve/20000.));
printf(" 0.60 - .64999          %f\n\n",(thirteen/20000.));
printf(" 0.65 - .69999          %f\n\n",(fourteen/20000.));
printf(" 0.70 - .74999          %f\n\n",(fifteen/20000.));
printf(" 0.75 - .79999          %f\n\n",(sixteen/20000.));
printf(" 0.80 - .84999          %f\n\n",(seventeen/20000.));
printf(" 0.85 - .89999          %f\n\n",(eighteen/20000.));
printf(" 0.90 - .94999          %f\n\n",(nineteen/20000.));
printf(" 0.95 - 1.0000          %f\n\n",(twenty/20000.));
}
```

Distribution of random numbers

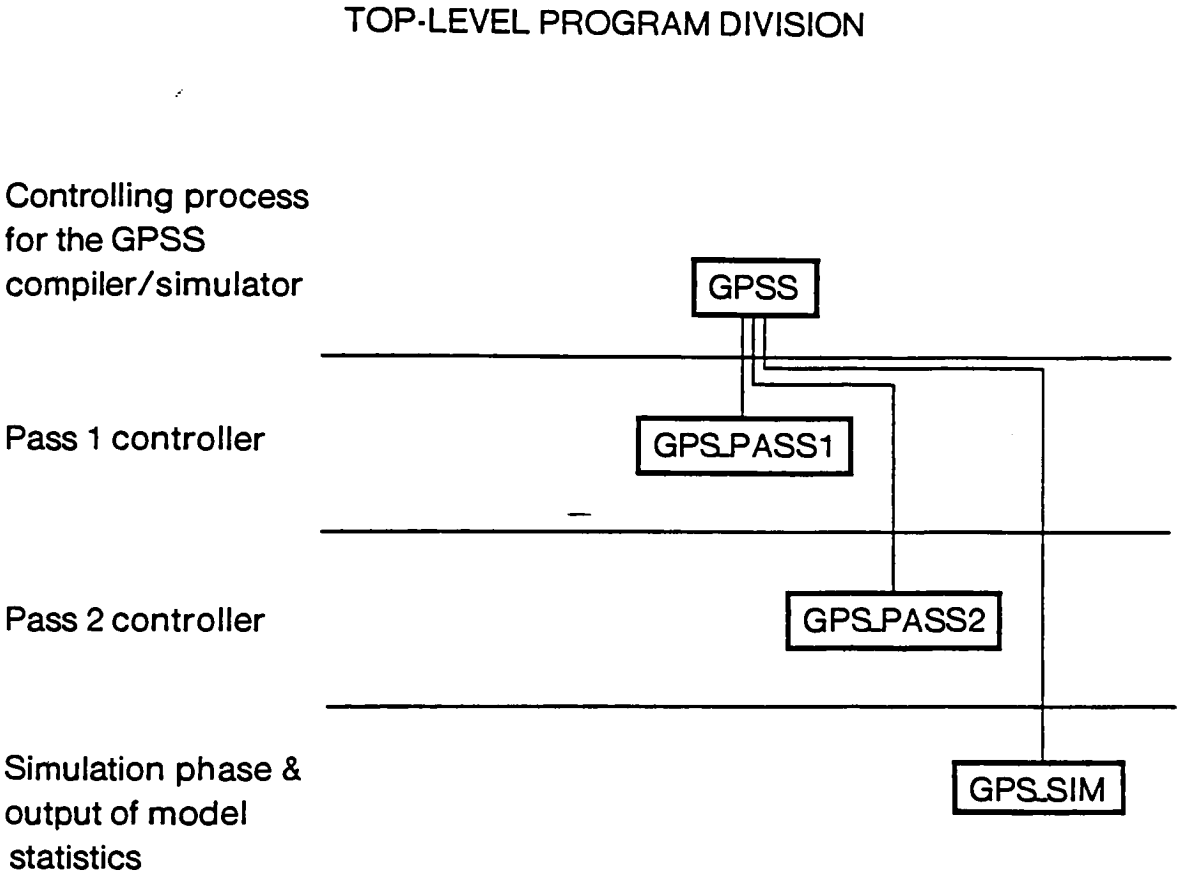
0.00 - .04999	0.050210
0.05 - .09999	0.049180
0.10 - .14999	0.049250
0.15 - .19999	0.049530
0.20 - .24999	0.050040
0.25 - .29999	0.048970
0.30 - .34999	0.050190
0.35 - .39999	0.050100
0.40 - .44999	0.048800
0.45 - .49999	0.050390
0.50 - .54999	0.050680
0.55 - .59999	0.050460
0.60 - .64999	0.050270
0.65 - .69999	0.049600
0.70 - .74999	0.050910
0.75 - .79999	0.049860
0.80 - .84999	0.051340
0.85 - .89999	0.050320
0.90 - .94999	0.049810
0.95 - 1.0000	0.050090

11.6. Appendix F - Command Definition File for GPSS command line

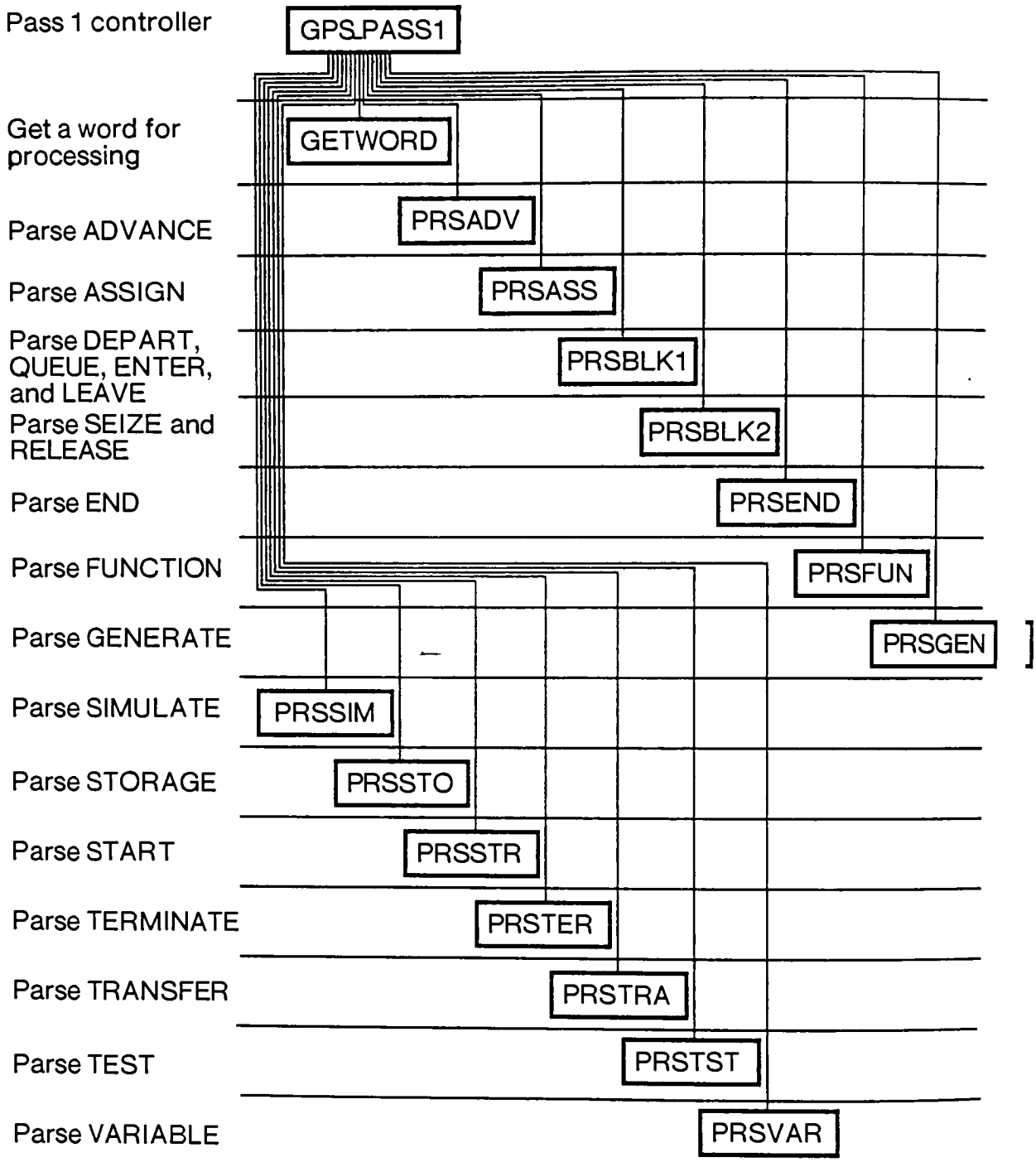
```
!*+
!* NAME:
!*              gpss.cld        - defines the command to start the GPSS compiler
!*                                /simulator
!* SYNOPSIS:
!*              set command gpss
!*
!*
!* PURPOSE:
!*              Defines the command to start the GPSS compiler/simulator
!*
!* RETURNS:
!*              Not applicable.
!*
!* MAINTENANCE:
!*              6-Jun-83        jvd     created
!*/

!
!       main definition of the verb GPSS
!
define verb gpss
    image USER$1:[vandellon.thesis]gpss
        parameter p1, label= filename, prompt= "filename", value(required)
```

11.7. Appendix G - GPSS module interconnection



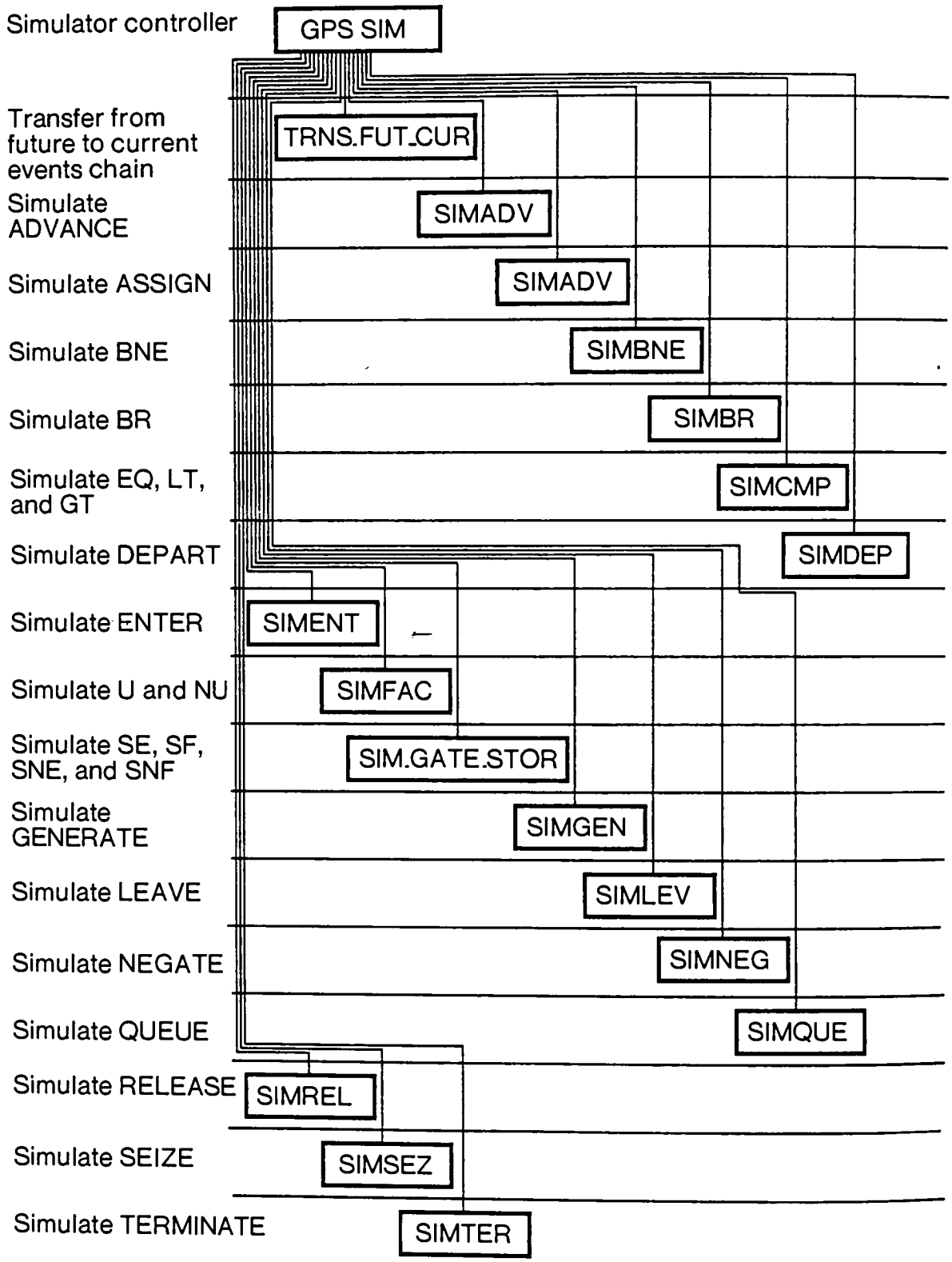
PASS 1 PROGRAM DIVISION



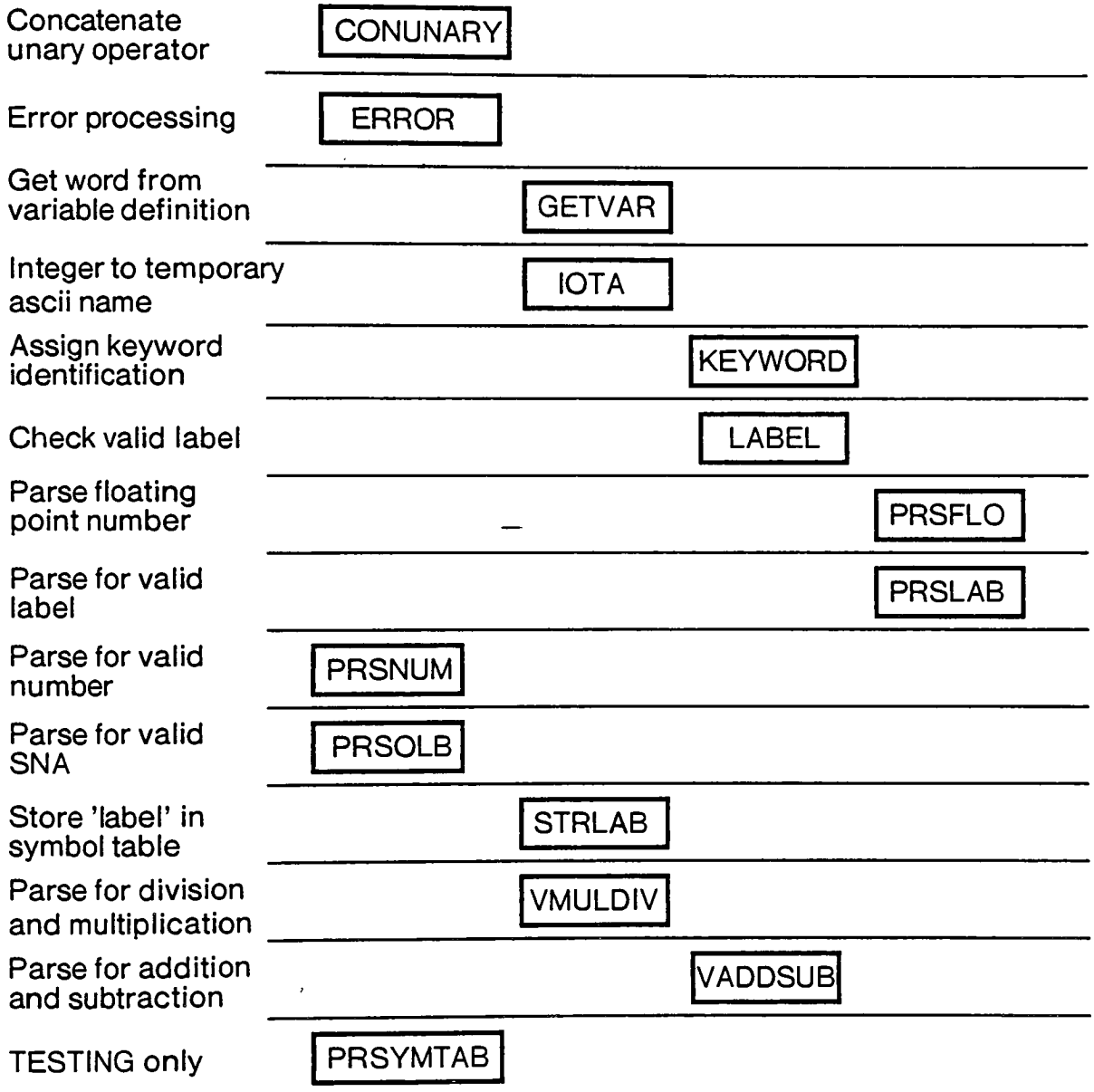
PASS 2 PROGRAM DIVISION

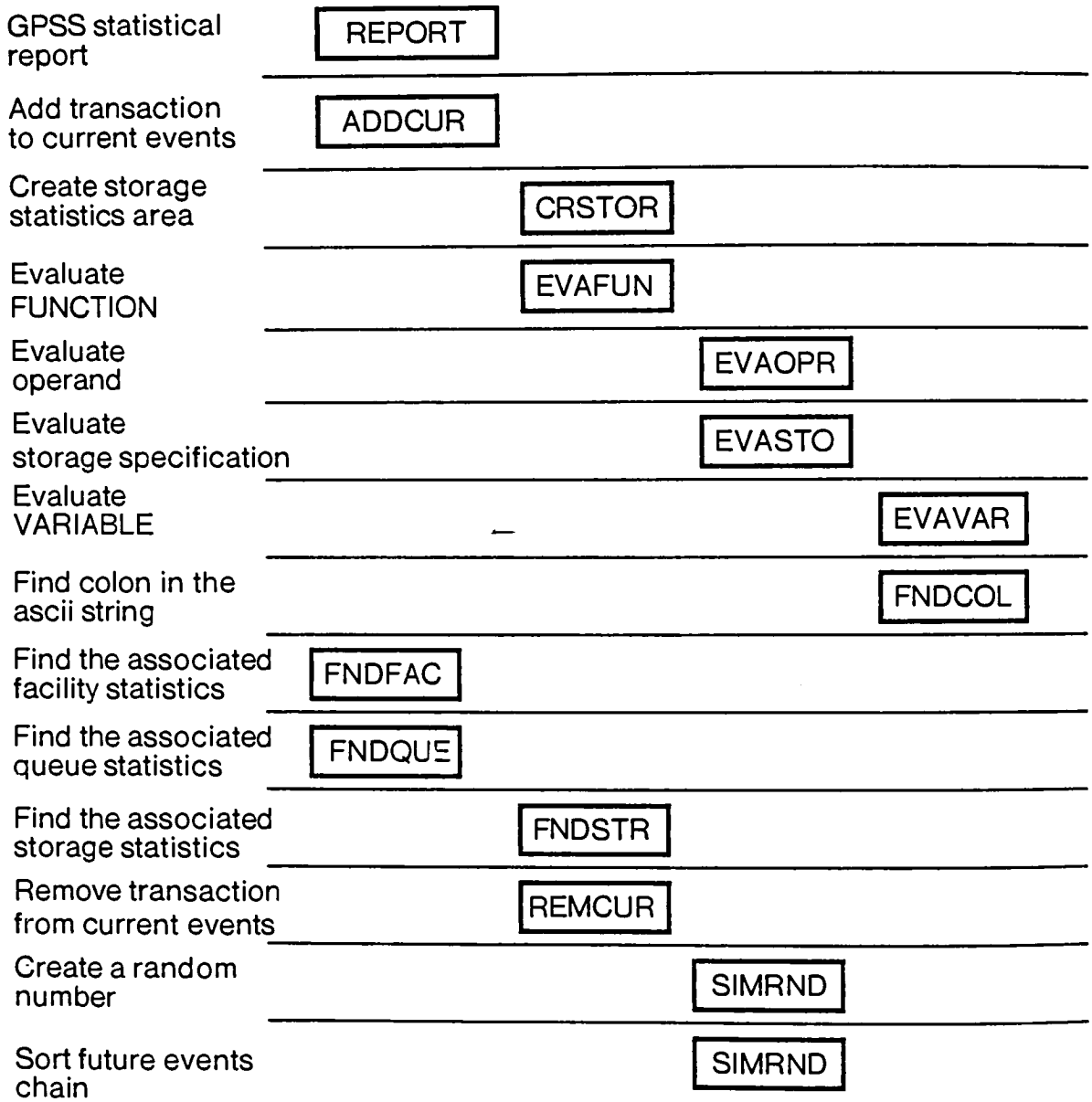
Pass 2 controller	GPS_PASS2
Tool to create skeleton pseudo code - used in Pass1	CREPSD
TESTING only	PS2TST

SIMULATOR PROGRAM DIVISION



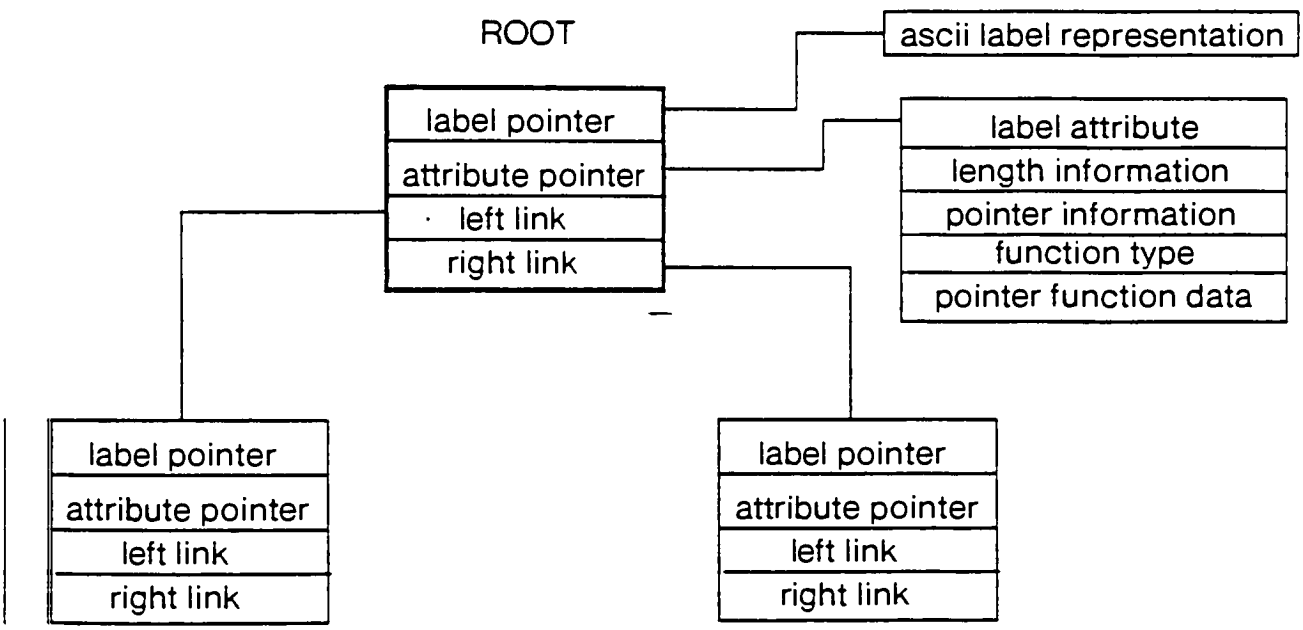
PASS 1 PROGRAM TOOLS



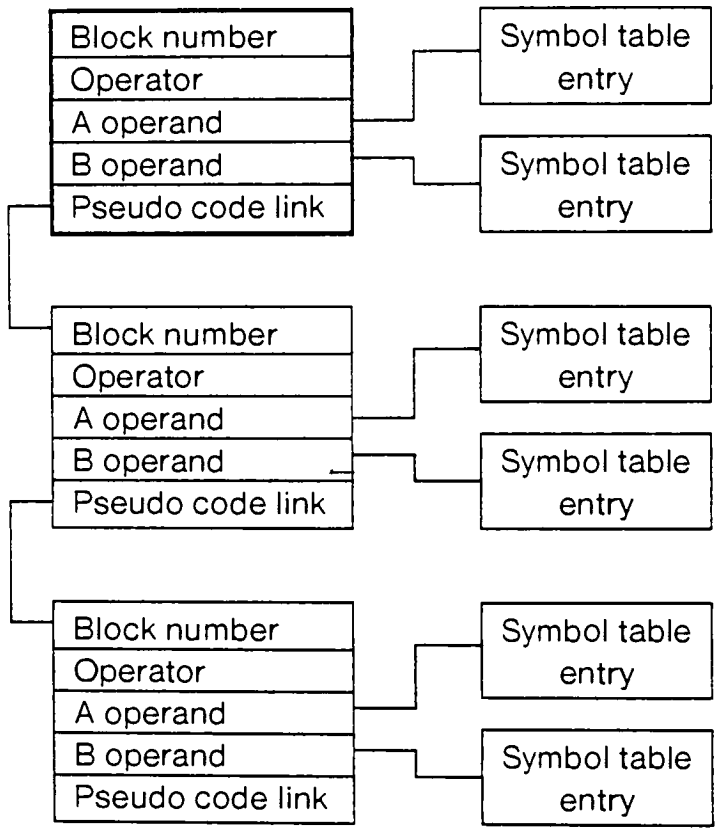

GPSS statistical report
REPORT
Add transaction to current events
ADDCUR
Create storage statistics area
CRSTOR
Evaluate FUNCTION
EVAFUN
Evaluate operand
EVAOPR
Evaluate storage specification
EVASTO
Evaluate VARIABLE
EVAVAR
Find colon in the ascii string
FNDCOL
Find the associated facility statistics
FNDFAC
Find the associated queue statistics
FNDQUE
Find the associated storage statistics
FNDSTR
Remove transaction from current events
REMCUR
Create a random number
SIMRND
Sort future events chain
SIMRND

11.8. Appendix H - GPSS internal architecture

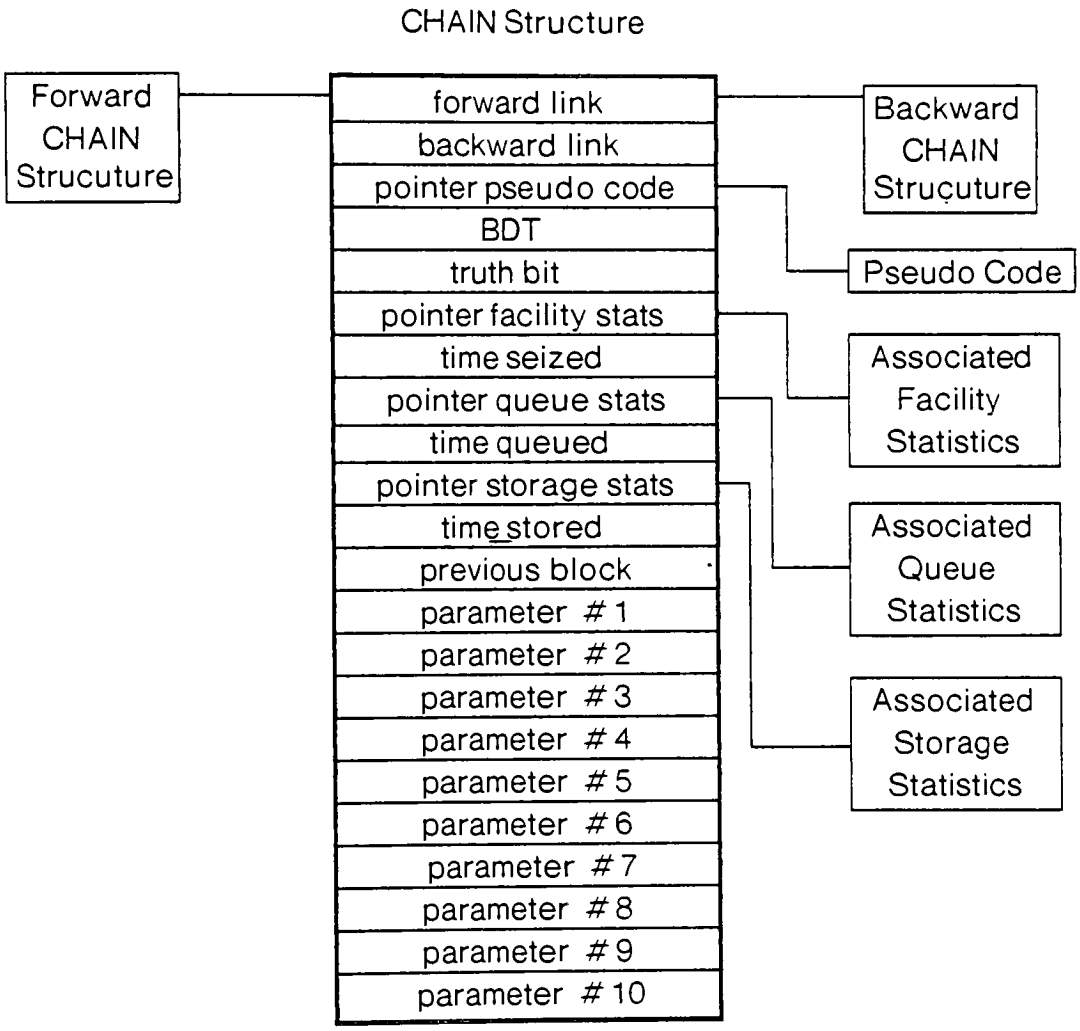
SYMBOL TABLE ARCHITECTURE



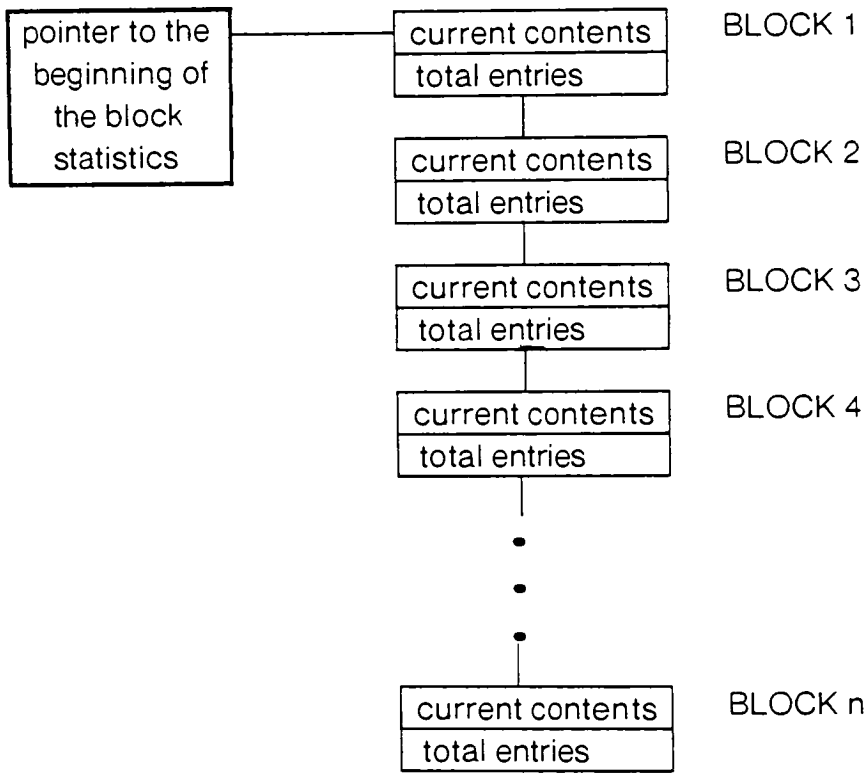
PSEUDO CODE ARCHITECTURE



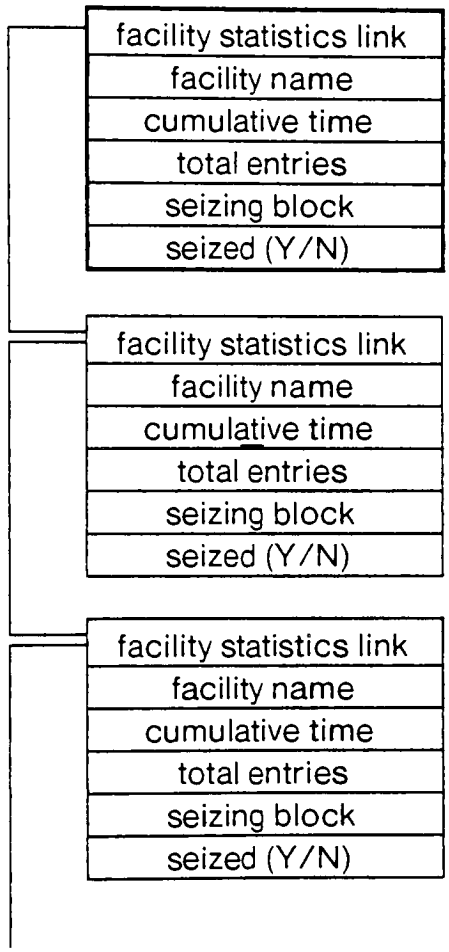
CURRENT and FUTURE EVENTS CHAIN
ARCHITECTURE



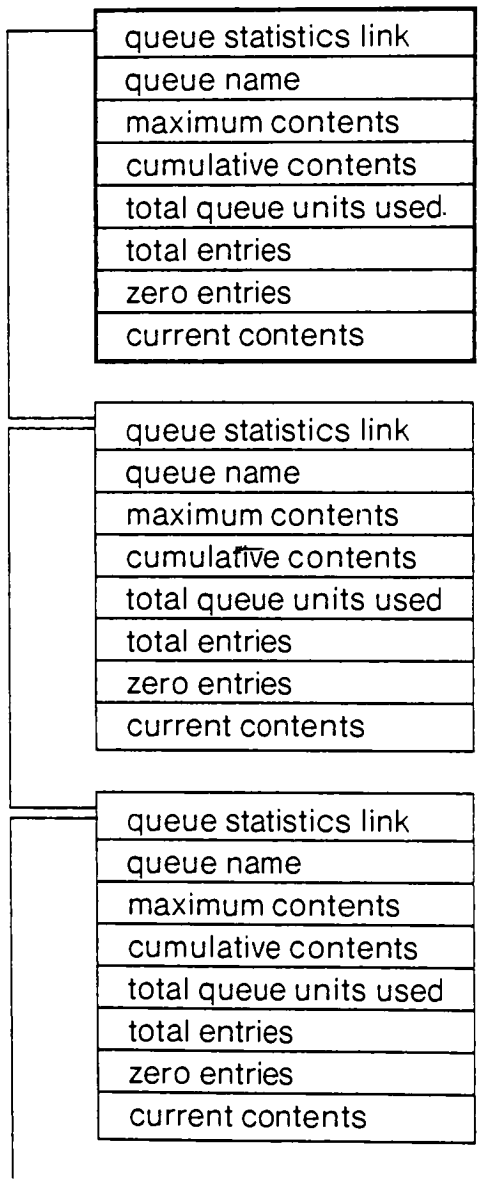
BLOCK STATISTICS ARCHITECTURE



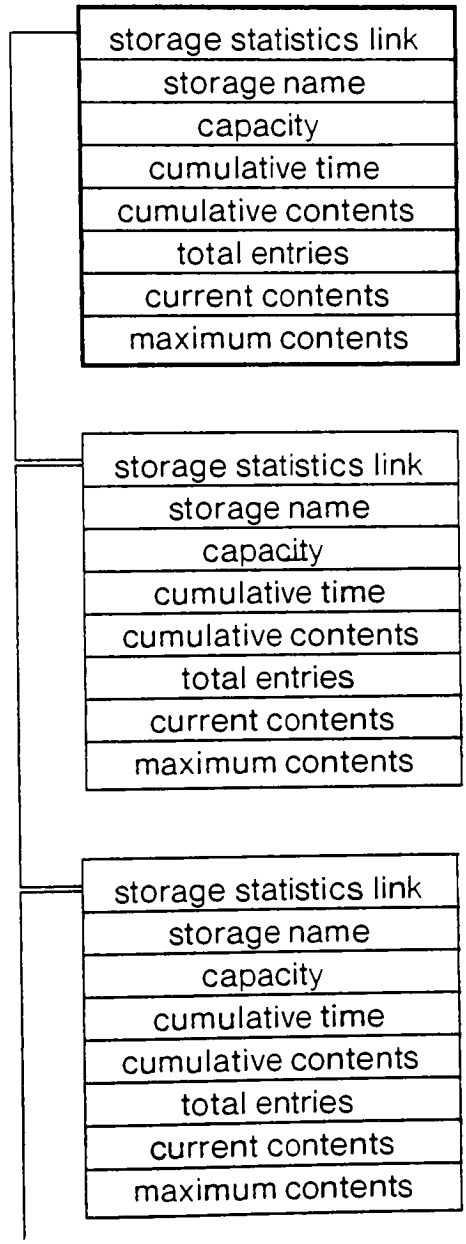
FACILITY STATISTICS ARCHITECTURE



QUEUE STATISTICS ARCHITECTURE



STORAGE STATISTICS ARCHITECTURE



12.1. Pass 1 and Tools Program Listings.

*** NOTE ***
The following pages are the program listings for the thesis project. Due to printing capabilities of the hardware, an underscore is an undefined character. The ← character is the representation for the underscore.

```
/*+
** NAME:
**              gpss
**
**
** SYNOPSIS:
**              main()
**
**
** PURPOSE:
**              Main routine for the GPSS compiler/simulator.  Starts and
**              controls everything.
**
** RETURNS:
**              Not applicable
**
**
** MAINTENANCE:
**              6-June-83       jvd     created
**
**
-*/
#include        stdio
#include        descrip
#include        ssdef

#include        "[vandellon.thesis]gpss"

globaldef       FILE    *fileptr= 0;                                    /* input file pointer *
**/
globaldef       int     line_number= 0;                                 /* line number of sourc
**e program */
globaldef       int     block_number= 0;                                /* block number in GPSS
** program */
globaldef       char    line[81]= 0;                                    /* area for source line
** */
globaldef       int     line_pointer= 0;                                /* beginning word point
**er */
globaldef       int     end_line_pointer= 0;                            /* end of word pointer
***/
globaldef       int     simulate_key= 0;                                /* encountered SIMULATE
** statement */
globaldef       int     end_key= 0;                                     /* encountered END stat
**ement */
globaldef       int     error_count= 0;                                 /* keep track of number
** of syntatic errors */
globaldef       INFO_STRT *strt[50]= 0;                                 /* keep track of start
**information */
globaldef       int     strt_ptr= 0;                                    /* keep track of the nu
**mber of start cards */
globaldef       int     temp_var;                                       /* the number of tempor
**ary labels */
globaldef       CODE    *ps_root;                                       /* first statement in p
**seudo code */
globaldef       CODE    *ps_cur;                                        /* last statement in ps
**eudo code or most current translation */

int main()
{
    int         ret_status;                                             /* generic return statu
**s */
    char        text_file_name[80]= 0;                                  /* GPSS filename */
    SYM_TABLE   *root;
    SYM_TABLE   *gps_pass1();

    $DESCRIPTOR(gpss_name,"filename");                                  /* tell system what par
**ameter */
    $DESCRIPTOR(file_name,text_file_name);                              /* tell system where to
** put parameter */

    if((ret_status= CLI$GET_VALUE(&gpss_name,&file_name)) != SS$_NORMAL)        /* use system to get in
** filename */
        exit(ret_status);
    else
        {
        printf("\n*     GPSS V1.0         May,1984\n*     author: J. Van Dellon\n");
```

```
        musnfl(text_file_name);                                         /* null terminate filen
**ame */
        if (( fileptr= fopen(text_file_name,"r")) == NULL)              /* open for read only *
**/
                {
                error(OPEN_ERR);
                error(FINISH_ERR);
                exit();
                }

        root= gps_pass1();
        gps_pass2(root);
/* check phase 2 */
/*ps2tst();*/
/* TEST ********************/

        if(simulate_key != ON)
                printf("\n>>> No SIMULATE statement encountered.  Execution halted");
        if(end_key != ON)
                error(NO_END);
        if(error_count > 0)
                error(FINISH_ERR);
        else
                gps_sim(root);

        fclose(fileptr);                                                /* close input file */
        }

}
```

```
/*+
** NAME:
**              gps-pass1
**
** SYNOPSIS:
**              SYM-TABLE *gps-pass1()
**
** PURPOSE:
**              Main routine for pass1 of the compiler.
**
** RETURNS:
**              root    - if no errors
**              -1      - if errors
**
** MAINTENANCE:
**              7-June-83       jvd     created
**
**
-*/
#include        stdio
#include        descrip
#include        ssdef
#include        "[vandellon.thesis]gpss"

globaldef       int     label-light= 0;
globaldef       SYM-TABLE *cur-label= 0;

globalref       FILE    *fileptr;
globalref       char    line[];
globalref       int     line-number;
globalref       int     block-number;
globalref       int     line-pointer;
globalref       int     end-line-pointer;
globalref       COOE    *ps-cur;
globalref       COOE    *ps-root;

int gps-pass1()
{
    int         command;
    int         err-flag= SUCCESS;
    char        *newword;
    char        *getword();
    int         keyword();
    SYM-TABLE   *root;

    line[80]= NULL;                                     /* initialize area for source line */
    root= malloc(sizeof(SYM-TABLE));                    /* get area for first entry into symbol
** table*/
    ps-root= malloc(sizeof(COOE));                      /* get area for first entry in pseudo c
**ode */
    ps-cur= ps-root;
    ps-cur->operator= NULL;
    root->label=NULL;                                   /* initialize blank entry in symbol tab
**le */

    while( fgets(line,MAXLINE,fileptr) != NULL )        /* get source line */
        {
        line-number++;                                  /* got another line */
        if( line[0] != '!')                             /* check for a comment line */
                {
                err-flag= line-pointer= end-line-pointer= label-light= 0;/* initialize for new line */
                while((((newword= getword()) != EOL) && (newword != COMMA)) && (err-flag != ERROR))
                        {
                        switch(command=  keyword(newword) )     /* find out whats on the line */
                                {
                                case(SIMULATE):                 /* parsing for the SIMULATE statement *
**/

                                        printf("%d              %s",line-number,line);/* print source l
**ine to user */
                                        err-flag= prssim(root);
                                        break;

                                case(START):                    /* parsing for the START statement */
```

```
**ine to user */
                        printf("%d              %s",line_number,line);/* print source
                        err_flag= prsstr(root);
                        break;

                case(END):                      /* parsing for the END statement */
**ine to user */
                        printf("%d              %s",line_number,line);/* print source
                        err_flag= prsend();
                        break;

                case(STORAGE):                  /* parsing for the STORAGE statement *
**
**ine to user */
                        printf("%d              %s",line_number,line);/* print source
                        err_flag= prssto(root);
                        break;

                case(VARIABLE):                 /* parsing for the VARIABLE statement
**/
**ine to user */
                        printf("%d              %s",line_number,line);/* print source
                        err_flag= prsvar(root);
                        break;

                case(FUNCTION):                 /* parsing for the FUNCTION statement
**/
**ine to user */
                        printf("%d              %s",line_number,line);/* print source
                        err_flag= prsfun(root);
                        break;

                case(GENERATE):                 /* parsing for the GENERATE statement
**/
                        block_number++;
                        printf("%d      %d      %s",line_number,block_number,line);/*
**rint source line to user */
                        err_flag= prsgen(root);
                        break;

                case(TERMINATE):                /* parsing for the TERMINATE statement
***/
                        block_number++;
                        printf("%d      %d      %s",line_number,block_number,line);/*
**rint source line to user */
                        err_flag= prster(root);
                        break;

                case(TRANSFER):                 /* parsing for the TRANSFER statement
**/
                        block_number++;
                        printf("%d      %d      %s",line_number,block_number,line);/*
'**rint source line to user */
                        err_flag= prstra(root);
                        break;

                case(ADVANCE):                  /* parsing for the ADVANCE statement *
**
                        block_number++;
                        printf("%d      %d      %s",line_number,block_number,line);/*
**rint source line to user */
                        err_flag= prsadv(root);
                        break;

                case(ASSIGN):                   /* parsing for the ASSIGN statement */
                        block_number++;
                        printf("%d      %d      %s",line_number,block_number,line);/*
```

```
**rint source line to user */
                                        err-flag= prsass(root);
                                        break;

                                case(SEIZE):                    /* parsing for the SEIZE & RELEASE stat
**ement */
                                case(RELEASE):

                                        block-number++;
                                        printf("%d      %d      %s",line-number,block-number,line);/* p
**rint source line to user */
                                        err-flag= prsblk2(root,command);
                                        break;

                                case(ENTER):
                                case(LEAVE):
                                case(QUEUE):
                                case(DEPART):                   /* parsing for the ENTER, LEAVE, QUEUE,
** & DEPART */

                                        block-number++;
                                        printf("%d      %d      %s",line-number,block-number,line);/* p
**rint source line to user */
                                        err-flag= prsblk1(root,command);
                                        break;

                                case(GATE):                     /* parsing for the GATE statement */

                                        block-number++;
                                        printf("%d      %d      %s",line-number,block-number,line);/* p
**rint source line to user */
                                        err-flag= prsgat(root);
                                        break;

                                case(TEST):                     /* parsing for the TEST statement */

                                        block-number++;
                                        printf("%d      %d      %s",line-number,block-number,line);/* p
**rint source line to user */
                                        err-flag= prstst(root);
                                        break;

                                default:                        /* means you got a label */
                                        if((err-flag= label(newword,root)) == SUCCESS)
                                                label-light= ON;
                                        break;
                                }
                        }

                if((newword == COMMA) && (err-flag != ERROR))
                        {
                        printf("%d              %s",line-number,line);
                        error(SYNTAX);
                        }
                }
        else
                printf("%d              %s",line-number,line);

        }

/*      prsymtab(root);                                                 print out contents of
**symbol table ***TEST*** */
    return(root);
}
```

```
/*+
** NAME:
**              prsadv
**
** SYNOPSIS:
**              int prsadv(root)
**                      SYM←TABLE       *root
**
** PURPOSE:
**              Parse operands for ADVANCE instruction.
**
** RETURNS:
**              ERROR   - syntatic error found
**              SUCCESS - OK
**
** MAINTENANCE:
**              15-June-83      jvd     created
**              18-Mar-84       jvd     added pseudo code generation
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       char    line[];
globalref       int     label←light;
globalref       SYM←TABLE *cur←label;
globalref       CODE    *ps←cur;
globalref       int     block←number;

int prsadv(root)
    SYM←TABLE   *root;

    {
    int         ret←status;
    char        *word;
    SYM←TABLE   *hold←label;
    int         prsolb();
    int         prsnum();

    if(prslab(hold←label) == ERROR)                         /* check for valid label */
        return(ERROR);

    crepsd(AOVANCE);                                        /* create pseudo code for block */
    ps←cur->block= block←number   1;

    if(((word= getword()) == EOL) || (word == COMMA))       /* get A-operand */
        return(error(REQ←OPER));

    if(isdigit(*word))                                      /* if numeric check for validity */
        {
        if(prsnum(word,root) == ERROR)
                return(error(NO←OPER));
        }
    else if(((ret←status= prsolb(word,root)) == ERROR) && (ret←status == LAB←STOR)
        & (ret←status == SNA←RANDOM))                       /* if alpha check for SNA */
        return(error(NO←OPER));

    ps←cur->A←operand= cur←label;
    cur←label= hold←label;

    if((word= getword()) != EOL)                            /* make sure thats all on the line */
        return(error(SYNTAX));

    ps←cur= ps←cur->code←link;
    return(SUCCESS);
    }
```

```
/*+
** NAME:
**              prsass
**
** SYNOPSIS:
**              int prsass(root)
**                      SYM←TABLE       *root;
**
** PURPOSE:
**              Parse operands for ASSIGN instruction.
**
** RETURNS:
**              ERROR   - any error found when parsing the ASSIGN statement
**
** MAINTENANCE:
**              15-June-83      jvd     created
**              18-Mar-84       jvd     added pseudo code generation
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       char    line[];
globalref       int     end←line←pointer;
globalref       SYM←TABLE *cur←label;
globalref       CODE    *ps←cur;
globalref       char    temp←label[];
globalref       int     block←number;

int prsass(root)
    SYM←TABLE   *root;
    {
    char        *word;
    SYM←TABLE   *hold←label;            /* hold cur←label address */
    SYM←TABLE   *hold←A;                /* hold A-oper if 2 pseudo code statements needed */
    SYM←TABLE   *hold←B;                /* hold B-oper if 2 pseudo code statements needed */
    int         ret←status;
    int         work←var= 1;            /* used if negative sign encountered */

    if(prslab(hold←label) == ERROR)                             /* check for correct label */
        return(ERROR);

    crepsd(ASSIGN);                                             /* create pseudo code for block */
    ps←cur->block= block←number   1;                            /* assign correct block id */

    if(((word= getword()) == EOL) || (word == COMMA))           /* get A-operand */
        return(error(REQ←OPER));

    if(isdigit(*word))                                          /* A-operand a numeric check for validi
**ty */
        {
        if(prsnum(word,root) == ERROR)
                return(ERROR);
        cur←label->attr←pnt->lab←attribute= SNA←PARAM;
        }
    else if(((ret←status= prsolb(word,root)) == ERROR) || (ret←status == LAB←STOR)
        || (ret←status == SNA←RANDOM))                          /* A-operand alpha check valid SNA */
        return(error(NO←OPER));

    ps←cur->A←operand= cur←label;                               /* assign A←operand */
    cur←label= hold←label;                                      /* get correct current label back */

    if((word= getword()) != COMMA)                              /* check for COMMA */
        return(error(REQ←OPER));

    if(((word= getword()) == EOL) || (word == COMMA))           /* get B-operand */
        return(error(SYNTAX));

    if((line[end←line←pointer] == '+') || (line[end←line←pointer] == '-'))/* check for correct operator
** */
        {
        if(line[end←line←pointer] == '-')                       /* subtraction arithmetic */
                work←var= -1;
        end←line←pointer++;                                     /* go to next
        if(((word= getword()) == EOL) || (word == COMMA))       /* get rest o
```

```
            return(error(SYNTAX));
        else if(isdigit(*word))                                 /* numeric check for validity */
                {
                if(prsnum(word,root) == ERROR)
                        return(ERROR);
                *(cur->label->attr->pnt->pnt->info) *= work->var;   /* change sign of numeric if negative *
**/
                ps->cur->B->operand= cur->label;                /* get B->oper in place */
                }
        else if(((ret->status= prsolb(word,root)) != ERROR) && (ret->status != LAB->STOR)
                && (ret->status != SNA->RANDOM))                /* alpha check for validity */
                {
                if(work->var > 0)                               /* has a negative sign been found */
                        ps->cur->B->operand= cur->label;        /* NO! */
                else
                        {
                        hold->A= ps->cur->A->operand;           /* store A-oper */
                        ps->cur->operator= NEG;                 /* add new pseudo code statement */
                        ps->cur->A->operand= cur->label;        /* put in A-oper */
                        temp->label[3]= NULL;
                        if(strlab(root,strcat(temp->label,itoa()),0) == ERROR)/* create new value label
***/
                                return(error(COMPILER));
                        hold->B= ps->cur->B->operand= cur->label;
                        cur->label->attr->pnt->lab->attribute= CONSTANT;
                        cur->label->attr->pnt->leng->info= 1;
                        cur->label->attr->pnt->pnt->info= malloc(sizeof(int));
                        ps->cur= ps->cur->code->link;           /* new pseudo code negates value */
                        crepsd(ASSIGN);                         /* create pseudo code for assign */
                        ps->cur->block= block->number - 1;
                        ps->cur->A->operand= hold->A;
                        ps->cur->B->operand= hold->B;
                        }
                }
        else
                return(error(NO->OPER));
        cur->label= hold->label;                                /* restore good label */
        }

    if(getword() != EOL)                                        /* make sure end of line */
        return(error(SYNTAX));

    ps->cur= ps->cur->code->link;                               /* get to new pseudo code statement */
}
```

```
/*+
** NAME:
**              prsblk1
**
** SYNOPSIS:
**              int prsblk1(root,command)
**                      SYM←TABLE       *root
**                      int             command
**
** PURPOSE:
**              Parse operands for DEPART, QUEUE, ENTER, and LEAVE instruction.
**
** RETURNS:
**              ERROR   - syntatic error detected
**              SUCCESS - OK
** MAINTENANCE:
**              15-June-83      jvd     created
**              18-Mar-84       jvd     added generation of pseudo code
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       char    line[];
globalref       SYM←TABLE *cur←label;
globalref       CODE    *ps←cur;
globalref       char    temp←label[];
globalref       int     block←number;

int prsblk1(root,command)
    SYM←TABLE   *root;
    int         command;
    {
    char        *word;
    SYM←TABLE   *hold←label= NULL;
    int         prsnum();
    int         ret←status;

    if(prslab(hold←label) == ERROR)                             /* check for valid label */
        return(ERROR);

    crepsd(command);                                            /* fill in pseudo code */
    ps←cur->block= block←number - 1;

    if(((word= getword()) == EOL) || (word == COMMA))           /* get A-operand */
        return(error(REQ←OPER));

    if(isdigit(*word))                                          /* check for valid numeric */
        {
        if(prsnum(word,root) == ERROR)
                return(ERROR);
        ps←cur->A←operand= cur←label;
        }
    else if((command == ENTER) || (command == LEAVE))
        {
        if(((ret←status= prsolb(word,root)) != ERROR) && (ret←status != SNA←RANDOM))/* check for valid
 **SNA */
                ps←cur->A←operand= cur←label;
        else
                return(error(NO←OPER));
        }
    else if((command == DEPART) || (command == QUEUE))
        {
        if(((ret←status= prsolb(word,root)) != ERROR) && (ret←status != LAB←STOR)/* check for valid SNA
 ** */
                && (ret←status != SNA←RANDOM))
                ps←cur->A←operand= cur←label;
        else
                return(error(NO←OPER));
        }
    cur←label= hold←label;

    if((word= getword()) == EOL)                                /* no B-opera
        {
```

```
    temp->label[3]= NULL;
    if(strlab(root,strcat(temp->label,itoa()),0) == ERROR)
            return(error(COMPILER));
    ps->cur->B->operand= cur->label;
    cur->label->attr->pnt->lab->attribute= CONSTANT;
    cur->label->attr->pnt->leng->info= 0;
    cur->label->attr->pnt->pnt->info= malloc(sizeof(int));
    *cur->label->attr->pnt->pnt->info= 1;
    ps->cur= ps->cur->code->link;
    return(SUCCESS);
    }
else if(word == COMMA)                                  /* else if COMMA look for B-operand */
    {
    if(((word= getword()) == EOL) || (word == COMMA))
            return(error(SYNTAX));
    }

if(isdigit(*word))                                      /* check for valid numeric */
    {
    if(prsnum(word,root) == ERROR)
            return(ERROR);
    }
else if((ret->status= prsolb(word,root) == ERROR) || (ret->status == LAB->STOR)
            || (ret->status == SNA->RANOOM))            /* check for valid SNA */
    return(error(NO->OPER));
ps->cur->B->operand= cur->label;
cur->label= hold->label;

if((word= getword()) != EOL)                            /* make sure end of line */
    return(error(SYNTAX));

ps->cur= ps->cur->code->link;                           /* get to new pseudo code statement */
}
```

```
/*+
** NAME:
**              prsblk2
**
** SYNOPSIS:
**              int prsblk2(root,command)
**                      SYM←TABLE       *root
**                      int             command
**
** PURPOSE:
**              Parse operands for SEIZE and RELEASE instruction.
**
** RETURNS:
**              ERROR   - syntatic error detected
**
** MAINTENANCE:
**              15-June-83      jvd     created
**              18-Mar-84       jvd     added generation of pseudo code
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       char    line[];
globalref       SYM←TABLE *cur←label;
globalref       CODE    *ps←cur;
globalref       int     block←number;

int prsblk2(root,command)
    SYM←TABLE   *root;
    int         command;
    {
    char        *word;
    SYM←TABLE   *hold←label;
    int         ret←status;
    int         prsnum();

    if(prslab(hold←label) == ERROR)                         /* check for correct label */
        return(ERROR);

    crepsd(command);                                        /* create pseudo code for block stateme
**nt */
    ps←cur->block= block←number - 1;

    if(((word= getword()) == EOL) || (word == COMMA))       /* get A-operand */
        return(error(REQ←OPER));

    if(isdigit(*word))                                      /* check for valid numeric */
        {
        if(prsnum(word,root) == ERROR)
                return(ERROR);
        else if((word= getword()) != EOL)
                return(error(SYNTAX));
        ps←cur->A←operand= cur←label;
        }
    else
        {
        if(((ret←status= prsolb(word,root)) != ERROR) &&
                (ret←status != SNA←RANDOM) && (ret←status != LAB←STOR))/* check for valid SNA */
                ps←cur->A←operand= cur←label;
        else
                return(error(NO←OPER));
        }
    cur←label= hold←label;

    if((word= getword()) != EOL)                            /* make sure that is all on the line */
**
        return(error(SYNTAX));

    ps←cur= ps←cur->code←link;                              /* go to new pseudo code statement */
    }
```

```
/*+
** NAME:
**              prsend
**
** SYNOPSIS:
**              int prsend()
**
** PURPOSE:
**              Parse operands for END instruction.
**
** RETURNS:
**              ERROR   - syntatic error detected
**
** MAINTENANCE:
**              15-June-83      jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       int     end_key;
globalref       int     end_line_pointer;
globalref       int     label_light;

int prsend()
{
    end_line_pointer++;

    if(label_light == ON)                                   /* labels not allowed on END */
        return(error(LA8_NOT_ALW));

    if(end_key == ON)                                       /* check for multiple END's */
        return(error(MULT_END));

    end_key= ON;                                           -/* turn end light found on */

    if(getword() != EOL)                                    /* no operands allowed */
        return(error(NO_OPER));

}
```

```
/*+
** NAME:
**              prsfun
**
** SYNOPSIS:
**              int prsfun(root)
**                      SYM←TABLE       *root
**
** PURPOSE:
**              Parse operands for FUNCTION instruction.
**
** RETURNS:
**              ERROR   - syntatic error detected
**
** MAINTENANCE:
**              15-June-83      jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       FILE            *fileptr;
globalref       SYM←TABLE       *cur←label;
globalref       char            line[];
globalref       int             line←number;
globalref       int             end←line←pointer;
globalref       int             line←pointer;

int prsfun(root)
    SYM←TABLE   *root;
    {
    char        *word;
    int         ret←status;
    int         i, j;
    int         temp;
    SYM←TABLE   *hold←label;
    int         prsflo();

    if(prslab(hold←label) == ERROR)                             /* check for valid label */
        return(ERROR);

    cur←label->attr←pnt->lab←attribute= LAB←FUNC;               /* put FUNCTION label in symbol table *
**/

    if(((word= getword()) == EOL) || (word == COMMA))           /* get the A-operand */
        return(error(REQ←OPER));

    if(strcmp(word,"r") != 0)                                   /* make sure A-operand random number */
**
        {
        if(strcmp(word,"R") != 0)
                return(error(NO←OPER));
        }

    if((word= getword()) == EOL)                                /* get comma in syntax */
        return(error(REQ←OPER));
'   else if(word != COMMA)
        return(error(NO←OPER));

    if(((word= getword()) == EOL) || (word == COMMA))           /* get B-operand */
        return(error(REQ←OPER));

    if((*word == 'D') || (*word == 'd'))                        /* only valid function is discrete */
        {
        word++;
        if(prsnum(word,FUNCTION) == ERROR)
                return(error(ILL←OPER));
        cur←label->attr←pnt->leng←info= cur←label->attr←pnt->func←spec= atoi(word);/* update info in sy
**mbol table */

        cur←label->attr←pnt->pnt←info= malloc(sizeof(int) * cur←label->attr←pnt->leng←info);
        cur←label->attr←pnt->flo←num= malloc(sizeof(double) * cur←label->attr←pnt->leng←info);
        }
```

```
    else
        return(error(ILL-OPER));

    if(getword() != EOL)                                    /* make sure nothing else on line */
        return(error(SYNTAX));

    if(fgets(line,MAXLINE,fileptr) != NULL)                 /* get function trailer card */
        {
        line-number++;
        end-line-pointer= line-pointer= 0;
        printf("%d\t    %s",line-number,line);              /* print trailer card */
        temp= cur-label->attr-pnt->leng-info;
        for(j= 0, i= cur-label->attr-pnt->leng-info ; i > 0 ; j++, i--)
                {
                if(prsflo(j) == ERROR)
                        return(ERROR);                      /* check for valid floating point */

                if((word= getword()) != COMMA)              /* check for COMMA */
                        return(error(SYNTAX));

                if(((word= getword()) == EOL) || (word == COMMA))/* get next number in function trailer
** */
                        return(error(SYNTAX));

                if(isdigit(*word))                          /* check for valid number */
                        {
                        if(prsnum(word,FUNCTION) == ERROR)
                                return(error(ILL-FUNC));
                        else
                                *((cur-label->attr-pnt->pnt-info) + j)= atoi(word);/* put info into sym
**bol table */
                        }
                else
                        return(error(MIS-FUNC));

                if((((word= getword()) == EOL) && ((i - 1) != 0)) ||
                        (word == COMMA))                    /* check for '/' between data or EOL */
**                                                          -
                        return(error(ILL-FUNC));
                else if(word != EOL)
                        {
                        if(line[end-line-pointer] != '/')
                                return(error(ILL-FUNC));
                        else
                                end-line-pointer++;
                        }
                }
        if(getword() != EOL)                                /* make sure no more on line */
                return(error(SYNTAX));
        }
    else
        return(error(PRE-END));                             /* no trailer card */
    }
```

```
/*+
** NAME:
**              prsgat
**
** SYNOPSIS:
**              int prsgat(root)
**                      SYM←TABLE       *root
**
** PURPOSE:
**              Parse operands for GATE instruction.
**
** RETURNS:
**              ERROR   - if error found in parsing GATE statement
**
** MAINTENANCE:
**              15-June-83      jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       SYM←TABLE *cur←label;
globalref       char    temp←label[];
globalref       CODE    *ps←cur;
globalref       int     block←number;

int prsgat(root)
    SYM←TABLE   *root;
    {
    char        *word;
    char        option[4];
    int         ret←status;
    SYM←TABLE   *hold←label;
    CODE        *back←link;

    if(prslab() == ERROR)                                       /* check for correct la
**bel */
        return(ERROR);
    hold←label= cur←label;
    crepsd(GATE);
    ps←cur->block= block←number - 1;
    ps←cur->B←operand= NULL;

    if(((word= getword()) == EOL) || (word == COMMA))           /* get A-operand */
        return(error(REQ←OPER));
    else if((strcmp(word,"U") != 0) && (strcmp(word,"NU") != 0) &&
                (strcmp(word,"SF") != 0) && (strcmp(word,"SE") != 0) &&
                (strcmp(word,"u") != 0) && (strcmp(word,"nu") != 0) &&
                (strcmp(word,"sf") != 0) && (strcmp(word,"se") != 0) &&
                (strcmp(word,"SNE") != 0) && (strcmp(word,"sne") != 0) &&
                (strcmp(word,"snf") != 0) && (strcmp(word,"SNF") != 0))     /* check for valid mneu
**monic */
        return(error(ILL←OPT));
    else
        {
        strcpy(option,word,3);                                  /* save option for late
**r use */
        option[3]= NULL;
        if((strcmp(word,"U") == 0) || (strcmp(word,"u") == 0))
                ps←cur->operator= U;
        else if((strcmp(word,"NU") == 0) || (strcmp(word,"nu") == 0))
                ps←cur->operator= NU;
        else if((strcmp(word,"SF") == 0) || (strcmp(word,"sf") == 0))
                ps←cur->operator= SF;
        else if((strcmp(word,"SE") == 0) || (strcmp(word,"se") == 0))
                ps←cur->operator= SE;
        else if((strcmp(word,"SNF") == 0) || (strcmp(word,"snf") == 0))
                ps←cur->operator= SNF;
        else if((strcmp(word,"SNE") == 0) || (strcmp(word,"sne") == 0))
                ps←cur->operator= SNE;
        }

   if(((word= getword()) == EOL) || (word == COMMA))            /* get B-operand */
```

```
        return(error(REQ←OPER));
    else if(isdigit(*word))                                          /* valid numeric */
        {
        if(prsnum(word,root) == ERROR)
                return(ERROR);
        }
    else if(((ret←status= prsolb(word,root)) == ERROR) || (ret←status == SNA←RANDOM))/* valid SNA */
        return(error(NO←OPER));

    if((ret←status == LAB←STOR) && (option[0] != 's'))               /* check for storage SN
**A and not storage mneumonic */
        return(error(NO←OPER));
    else if((option[0] == 's') && (ret←status != LAB←STOR))          /* check for storage mn
**eumonic and not storage SNA */
        return(error(NO←OPER));

    ps←cur->A←operand= cur←label;
    cur←label= hold←label;
    back←link= ps←cur;
    ps←cur= ps←cur->code←link;
    crepsd(BNE);
    ps←cur->block= block←number - 1;
    ps←cur->B←operand= NULL;

    if((word= getword()) == EOL)                                     /* ok if no C-operand *
**/
        {
        temp←label[3]= NULL;
        if((ret←status= strlab(root,strcat(temp←label,itoa()),0)) == ERROR)
                return(error(COMPILER));
        ps←cur->A←operand= cur←label;
        cur←label->attr←pnt->lab←attribute= LAB←LABEL;
        cur←label->attr←pnt->leng←info= 1;
        cur←label->attr←pnt->pnt←info= malloc(sizeof(int));
        cur←label->attr←pnt->pnt←info= back←link;
        ps←cur= ps←cur->code←link;
        return(SUCCESS);
        }
    else if(word != COMMA)                                           /* check for special ch
**ar COMMA */
        return(error(SYNTAX));

    if(((word= getword()) == EOL) || (word == COMMA))                /* get C-operand */
        return(error(REQ←OPER));
    else if(isdigit(*word))                                          /* check for alpha, lab
**el */
        return(error(ILL←LAB));

    if((ret←status= strlab(root,word,SEARCH)) == ERROR)
        {
        ps←cur->A←operand= malloc(sizeof(SYM←TABLE));
        ps←cur->A←operand->label= malloc(strlen(word) + 1);
        strcpy(ps←cur->A←operand->label,word);
        *(ps←cur->A←operand->label + (strlen(word) + 1))= NULL;
        ps←cur->A←operand->attr←pnt= LAB←LABEL;
        }
    else
        ps←cur->A←operand= ret←status;

    if(getword() != EOL)                                             /* anything more on lin
**e error */
        return(error(SYNTAX));

    ps←cur= ps←cur->code←link;
    }
```

```
/*+
** NAME:
**              prsgen
**
** SYNOPSIS:
**              int prsgen(root)
**                      SYM←TABLE *root;
**
** PURPOSE:
**              Parse operands for GENERATE instruction.
**
** RETURNS:
**              ERROR   - syntatic error detected
**
** MAINTENANCE:
**              15-June-83      jvd     created
**              18-Mar-84       jvd     added pseudo code generation
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       CODE    *ps←cur;
globalref       SYM←TABLE *cur←label;
globalref       int     block←number;

int prsgen(root)
    SYM←TABLE   *root;
    {
    char        *word;
    SYM←TABLE   *hold←label= NULL;
    int         ret←status;

    if(prslab(hold←label) == ERROR)                         /* check for correct label */
        return(ERROR);

    crepsd(GENERATE);                                       /* create pseudo translation */
    ps←cur->block= block←number - 1;

    if(((word= getword()) == EOL) || (word == COMMA))       /* get A-operand */
        return(error(REQ←OPER));

    if(isdigit(*word))                                      /* check for numeric */
        {
        if(prsnum(word,root) == ERROR)
                return(ERROR);
        ps←cur->A←operand= cur←label;
        }
    else if(((ret←status= prsolb(word,root)) != ERROR) && (ret←status != LAB←STOR)
        && (ret←status != SNA←RANDOM) && (ret←status != SNA←PARAM))/* check for legal SNA */
        ps←cur->A←operand= cur←label;
    else
        return(error(NO←OPER));

    cur←label= hold←label;

    if((word = getword()) == EOL)                           /* get B-operand */
        {
        ps←cur->B←operand= NULL;                            /* no 8←operand */
        ps←cur= ps←cur->code←link;                          /* get ready for next translation */
        return(SUCCESS);                                    /* B-operand not required */
        }
    else if(word != COMMA)
        return(error(SYNTAX));

    if(((word= getword()) == EOL) || (word == COMMA))
        return(error(SYNTAX));

    if(isdigit(*word))                                      /* check for numeric */
        {
        if(prsnum(word,root) == ERROR)
                return(ERROR);
        ps←cur->B←operand= cur←label;
        }
```

```
else if(((ret←status= prsolb(word,root)) != ERROR) && (ret←status != LAB←STOR)
    && (ret←status != SNA←RANDOM) && (ret←status != SNA←PARAM))/* check for legal SNA */
    ps←cur->B←operand= cur←label;
else
    return(error(NO←OPER));

cur←label= hold←label;

if(getword() != EOL)                                    /* anything else on line */
    return(error(SYNTAX));

ps←cur= ps←cur->code←link;                              /* get ready for next translation */
}
```

```
/*+
** NAME:
**              prssim
**
** SYNOPSIS:
**              int prssim()
**
** PURPOSE:
**              Parse SIMULATE operands.
**
** RETURNS:
**              ERROR   - syntatic error detected
**
** MAINTENANCE:
**              15-June-83      jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       int     simulate←key;
globalref       int     label←light;

int prssim()
    {
    char        *word;

    if(label←light == ON)                                   /* label not allowed */
        return(error(LA8←NOT←ALW));

    if(simulate←key == ON)                                  /* check for multiple SIMULATE statemen
**ts */
        return(error(MULT←SIM));

    simulate←key= ON;                                       /* turn on SIMULATE found */

    if((word= getword()) == EOL)                            /* no oeprands allowed on SIMULATE */
        return(SUCCESS);
    else if(word == COMMA)
        return(error(SYNTAX));
    else
        return(error(NO←OPER));

    }
```

```
/*+
** NAME:
**              prssto
**
** SYNOPSIS:
**              int prssto(line)
**                      char    *line;
**
** PURPOSE:
**              Parse operands for STORAGE instruction.
**
** RETURNS:
**              Unknown
**
** MAINTENANCE:
**              15-June-83      jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       int     label←light;
globalref       SYM←TABLE *cur←label;

int prssto(root)
    SYM←TABLE   *root;
    {
    SYM←TABLE   *ret←status;
    int         number;
    char        *word;

    if(label←light != ON)                                       /* label reuired */
        return(error(LAB←REQ));

    cur←label->attr←pnt->lab←attribute= LAB←STOR;               /* put info into symbol table */

    if(isdigit(cur←label->label[0]))                            /* is current label a digit, legal */
        {
        if(prsnum(cur←label->label,STORAGE) == ERROR)
                return(error(ILL←LAB));
        }

    if(((word= getword()) == EOL) || (word == COMMA))           /* get A-operand */
        return(error(REQ←OPER));
    else if(isdigit(*word))                                     /* valid numeric */
        {
        number= atoi(word);
        cur←label->attr←pnt->leng←info - cur←label->attr←pnt->func←spec= number;/* tell amount of stora
**ge */
        cur←label->attr←pnt->pnt←info = malloc(number);         /* get a place for storage */
        }
    else
        return(error(NO←OPER));                                 /* Operand is not valid in this version
** unless numeric */

    if((word= getword()) != EOL)                                /* anything else on the line */
        return(error(SYNTAX));
    }
```

```
/*+
** NAME:
**              prsstr
**
** SYNOPSIS:
**              int prsstr(line)
**                      char    *line;
**
** PURPOSE:
**              Parse operands for START instruction.
**
** RETURNS:
**              ERROR   - syntatic error detected
**
** MAINTENANCE:
**              15-June-83      jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       int     label←light;
globalref       INFO←STRT *strt[];
globalref       int     strt←ptr;

int prsstr(root)
    SYM←TABLE   *root;
{
    char        *word;
    SYM←TA8LE   *ret←status;
    int         number;

    if(label←light == ON)                                       /* label not allowed */
        return(error(LAB←NOT←ALW));

    if(((word= getword()) == EOL) || (word == COMMA))           /* get A-operand */
        return(error(REQ←OPER));
    else if(isalpha(*word))                                     /* set up for assignment values (NOT U
**ED) */
        {
        if((ret←status= strlab(root,word,SEARCH)) == ERROR)
                return(error(UN←OPER));
        else if(ret←status->attr←pnt->lab←attribute != LAB←ASSIGN)
                return(error(NO←OPER));
        else
                {
                if(strt←ptr == 49)
                        return(error(TOO←START));
                strt[strt←ptr]= malloc(sizeof(INFO←STRT));
                strt[strt←ptr]->sim←time= *(ret←status->attr←pnt->pnt←info);
                }
        }
    else if(isdigit(*word))                                     /* check for valid numeric */
        {
        strt[strt←ptr]= malloc(sizeof(INFO←STRT));
        number= atoi(word);
        strt[strt←ptr]->sim←time= number;
        }

    if((word= getword()) == EOL)                                /* get B-operand */
        {
        (strt[strt←ptr])->print= PRT;
        strt←ptr++;
        return(SUCCESS);
        }
    else if(word == COMMA)                                      /* comma there */
        {
        if((word= getword()) == EOL)                            /* get C-operand */
                return(error(SYNTAX));
        }
    else
        return(error(SYNTAX));
```

```
    if(word == COMMA)                                          /* fill in snap print count time */
        (strt[strt-ptr])->print= PRT;
    else if((strcmp(word,"NULL") == 0) || (strcmp(word,"null") == 0))
        (strt[strt-ptr])->print= PRT;
    else if((strcmp(word,"NP") == 0) || (strcmp(word,"np") == 0))
        (strt[strt-ptr])->print= NP;
    else
        return(error(NO-OPER));

    if((word= getword()) == EOL)
        return(SUCCESS);
    else if(word == COMMA)
        {
        if((word= getword()) == EOL)
                return(error(SYNTAX));
        }

    if(isalpha(*word))                                         /* set up for assignment values (NOT US
**ED) */
        {
        if((ret-status= strlab(root,word,SEARCH)) == ERROR)
                return(error(UN-OPER));
        else if(ret-status != LAB-ASSIGN)
                return(error(NO-OPER));
        else
                ;
        }
    else if(isdigit(*word))
        {
        number= atoi(word);
        strt[strt-ptr]->interval-print= number;
        }
    else if(*word == NULL)
        strt[strt-ptr]->interval-print= 0;

    strt-ptr++;
}
```

```
/*+
** NAME:
**              prster
**
** SYNOPSIS:
**              int prster(root)
**
** PURPOSE:
**              Parse operands for TERMINATE instruction.
**
** RETURNS:
**              Unknown
**
** MAINTENANCE:
**              15-June-83      jvd     created
**              18-Mar-84       jvd     added pseudo code generation
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       CODE    *ps←cur;
globalref       SYM←TABLE *cur←label;
globalref       int     block←number;

int prster(root)
    SYM←TABLE   *root;
    {
    char        *word;
    SYM←TABLE   *hold←label= NULL;
    int         ret←status;
    int         prsnum();

    if(prslab(hold←label) == ERROR)                     /* check for valid label */
        return(ERROR);

    crepsd(TERMINATE);                                  /* create pseudo code structure */
    ps←cur->block= block←number   1;
    ps←cur->B←operand= NULL;                            /* no B-operand */

    if((word= getword()) == EOL)                        /* ok if nothing else on line */
        {
        if(prsnum("1",root) == ERROR)                   /* put in default of 1 for terminate s
**atement */
                return(error(SYNTAX));
        ps←cur->A←operand= cur←label;
        ps←cur= ps←cur->code←link;                      /* get ready for next line of code */
        return(SUCCESS);
        }
    else if(word == COMMA)                              /* COMMA NO!NO! */
        return(error(SYNTAX));
    else
        {
        if(isdigit(*word))                              /* valid numeric */
                {
                if(prsnum(word,root) == ERROR)
                        return(ERROR);
                ps←cur->A←operand= cur←label;           /* put info into pseudo code */
                }
        else
                return(error(NO←OPER));
        }

    cur←label= hold←label;

    if((word= getword()) != EOL)                        /* anything else on the line */
        return(error(SYNTAX));

    ps←cur= ps←cur->code←link;                          /* get ready for next line of code */
    }
```

```
/*+
** NAME:
**              prster
**
** SYNOPSIS:
**              int prster(root)
**
** PURPOSE:
**              Parse operands for TERMINATE instruction.
**
** RETURNS:
**              Unknown
**
** MAINTENANCE:
**              15-June-83      jvd     created
**              18-Mar-84       jvd     added pseudo code generation
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       CODE    *ps-cur;
globalref       SYM-TABLE *cur-label;
globalref       int     block-number;

int prster(root)
    SYM-TABLE   *root;
    {
    char        *word;
    SYM-TABLE   *hold-label= NULL;
    int         ret-status;
    int         prsnum();

    if(prslab(hold-label) == ERROR)                         /* check for valid label */
        return(ERROR);

    crepsd(TERMINATE);                                      /* create pseudo code structure */
    ps-cur->block= block-number - 1;
    ps-cur->B-operand= NULL;                                /* no B-operand */

    if((word= getword()) == EOL)                            /* ok if nothing else on line */
        {
        ps-cur->A-operand= NULL;
        ps-cur= ps-cur->code-link;                          /* get ready for next line of code */
        return(SUCCESS);
        }
    else if(word == COMMA)                                  /* COMMA NO!NO! */
        return(error(SYNTAX));
    else
        {
        if(isdigit(*word))                                  /* valid numeric */
                {
                if(prsnum(word,root) == ERROR)
                        return(ERROR);
                ps-cur->A-operand= cur-label;               /* put info into pseudo code */
                }
        else
                return(error(NO-OPER));
        }

    cur-label= hold-label;

    if((word= getword()) != EOL)                            /* anything else on the line */
        return(error(SYNTAX));

    ps-cur= ps-cur->code-link;                              /* get ready for next line of code */
    }
```

```
/*+
** NAME:
**              prstst
**
** SYNOPSIS:
**              int prstst(root)
**                      SYM←TABLE       *root;
** PURPOSE:
**              Parse operands for TEST instruction.
**
** RETURNS:
**              ERROR   - error when parsing the TEST statement
**
** MAINTENANCE:
**              15-June-83      jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       SYM←TABLE *cur←label;
globalref       CODE    *ps←cur;
globalref       char    temp←label[];
globalref       int     block←number;

int prstst(root)
    SYM←TABLE   *root;
    {
    char        *word;
    int         ret←status;
    SYM←TABLE   *hold←label;
    CODE        *back←link;

    if(prslab() == ERROR)                                   /* check for valid label */
        return(ERROR);

    crepsd(TEST);
    ps←cur->block= block←number - 1;
    hold←label= cur←label;

    if(((word= getword()) == EOL) || (word == COMMA))       /* get mneumonic option */
        return(error(REQ←OPER));

    if((strcmp(word,"E") != 0) && (strcmp(word,"e") != 0) &&    /* test for legal option */
     (strcmp(word,"L") != 0) && (strcmp(word,"l") != 0) &&
     (strcmp(word,"G") != 0) && (strcmp(word,"g") != 0))
        return(error(ILL←OPT));
    else if((strcmp(word,"E") == 0) || (strcmp(word,"e") == 0))
        ps←cur->operator= EQ;
    else if((strcmp(word,"L") == 0) || (strcmp(word,"l") == 0))
        ps←cur->operator= LT;
    else if((strcmp(word,"G") == 0) || (strcmp(word,"g") == 0))
        ps←cur->operator= GT;

    if(((word= getword()) == EOL) || (word == COMMA))       /* get B-operand, first SNA to be compa
**red */
        return(error(REQ←OPER));

    if(isdigit(*word))                                      /* check for numeric B-operand */
        {
        if(prsnum(word,root) == ERROR)
                return(ERROR);
        }
    else if(((ret←status= prsolb(word,root)) == ERROR) || (ret←status == LAB←STOR)
        || (ret←status == SNA←RANDOM))                      /* check for valid SNA */
        return(error(NO←OPER));
    ps←cur->A←operand= cur←label;
    cur←label= hold←label;

    if(((word= getword()) == EOL) || (word != COMMA))       /* check for COMMA */
        return(error(SYNTAX));

    if(((word= getword()) == EOL) || (word == COMMA))       /* get C-operand, second SNA to be compa
```

```
** */
        return(error(REQ-OPER));

    if(isdigit(*word))                                          /* check for numeric operand */
        {
        if(prsnum(word,root) == ERROR)
                return(ERROR);
        }
    else if(((ret-status= prsolb(word,root)) == ERROR) || (ret-status == LAB-STOR)
        || (ret-status == SNA-RANDOM))                          /* check for valid SNA */
        return(error(NO-OPER));
    ps-cur->B-operand= cur-label;
    cur-label= hold-label;

    back-link= ps-cur;
    ps-cur= ps-cur->code-link;
    crepsd(TEST);
    ps-cur->block= block-number - 1;
    ps-cur->B-operand= NULL;

    if(((word= getword()) == EOL) || (word != COMMA))           /* check for optional comma */
        {
        ps-cur->operator= BNE;
        temp-label[3]= NULL;
        if((ret-status= strlab(root,strcat(temp-label,itoa()),0)) == ERROR)
                return(error(COMPILER));
        ps-cur->A-operand= cur-label;
        cur-label->attr-pnt->lab-attribute= LAB-LABEL;
        cur-label->attr-pnt->leng-info= 1;
        cur-label->attr-pnt->pnt-info= malloc(sizeof(int));
        cur-label->attr-pnt->pnt-info= back-link;
        ps-cur= ps-cur->code-link;
        return(SUCCESS);
        }

    if(((word= getword()) == EOL) || (word == COMMA))           /* get D-operand, which is a label */
        return(error(SYNTAX));

    if(isdigit(*word))
        return(error(ILL-LAB));

    ps-cur->operator= BNE;
    if((ret-status= strlab(root,word,SEARCH)) == ERROR)
        {
        ps-cur->A-operand= malloc(sizeof(SYM-TABLE));
        ps-cur->A-operand->label= malloc(strlen(word) + 1);
        strcpy(ps-cur->A-operand->label,word);
        *(ps-cur->A-operand->label + (strlen(word) + 1))= NULL;
        ps-cur->A-operand->attr-pnt= LAB-LABEL;
        }
    else
        ps-cur->A-operand= ret-status;

    if((word= getword()) != EOL)                                /* anything else on the line */
        return(error(SYNTAX));

    ps-cur= ps-cur->code-link;
    }
```

```
/*+
** NAME:
**              prsvar
**
** SYNOPSIS:
**              int prsvar(root)
**                      SYM←TABLE       *root;
**
** PURPOSE:
**              Parse operands for VARIABLE instruction.
**
** RETURNS:
**              ERROR   - syntactic error detected
**
** MAINTENANCE:
**              21-Jan-84       jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       int     label←light;
globalref       SYM←TABLE *cur←label;
globalref       char    line[];
globaldef       int     unary= 0;

int prsvar(root)
    SYM←TABLE   *root;
    {
    char        *word;
    char        *strip←pnt;
    SYM←TABLE   *hold←label;

    unary= 0;
    hold←label= cur←label;
    if(label←light != ON)                                   /* check for required label */
        return(error(LAB←REQ));
    else
        {
        if(prslab() == ERROR)                               /* check for valid label */
                return(ERROR);
        cur←label->attr←pnt->lab←attribute= LAB←VAR;
        }

    cur←label->attr←pnt->pnt←info= malloc(sizeof(int) * 80);    /* allocate space for postfix notation
***/
    *cur←label->attr←pnt->pnt←info= NULL;

    if((word= getvar()) == EOL)                             /* get first value on line */
        return(error(REQ←OPER));
    else if(word == ERROR)
        return(ERROR);
    else if((word == DIVISION) || (word == MULTIPLICATION) || (word == ADDITION))/* if *, /. or + firs
** thing ERROR */
        return(error(ILL←VAR←DEF));
    else if(word == SUBTRACTION)                            /* unary '-' operator found */
        {
        unary= YES;                                         /* set negative unary flag */
        if((word= getvar()) == EOL)                         /* get a number or variable */
                return(error(REQ←OPER));                    /* if none there ERROR */
        else if(word == ERROR)
                return(ERROR);
        else if((word == ADDITION) || (word == SUBTRACTION) || (word == MULTIPLICATION)
                || (word == DIVISION))                      /* more operators are you nuts ERROR *
**
                return(error(ILL←VAR←DEF));
        }

    if(isdigit(*word))                                      /* if you have a number make sure val
** */
        {
        if(prsnum(word,VARIABLE) == ERROR)
                return(ERROR);
        }
```

```
else if(prsolb(word,root) != ERROR)                    /* if an SNA make sure valid */
    {
    strip-pnt= strpbrk(word,"*");
    strcpy(word,(strip-pnt+1));
    }
else
    return(ERROR);

cur-label= hold-label;

strcat(cur-label->attr-pnt->pnt-info,word);            /* put first operand out */
strcat(cur-label->attr-pnt->pnt-info,":");             /* terminator */
if(unary == YES)                                       /* if unary operator add to operand */
    conunary();

while((word= getvar()) != EOL)
    {
    if(word == ERROR)
            break;

    if((word == ADDITION) || (word == SUBTRACTION))
            {
            if(vaddsub(word,root) == ERROR)
                    {
                    word= ERROR;
                    break;
                    }
            }
    else if((word == MULTIPLICATION) || (word == DIVISION))
            {
            if(vmuldiv(word,root) == ERROR)
                    {
                    word= ERROR;
                    break;
                    }
            }
    else
            return(error(NO-OPER));

    cur-label->attr-pnt->leng-info= strlen(cur-label->attr-pnt->pnt-info);
    }

cur-label->attr-pnt->leng-info= strlen(cur-label->attr-pnt->pnt-info);
if(word == ERROR)
    return(ERROR);
else
    return(SUCCESS);
}
```

```
/*+
** NAME:
**              conunary
**
** SYNOPSIS:
**              int conunary()
**
** PURPOSE:
**              Concatinate unary operator '-' onto the current variable list.
**
** RETURNS:
**
** MAINTENANCE:
**              19-Feb-84       jvd     created
**
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"

globalref       SYM-TABLE *cur-label;
globalref       int     unary;

int conunary()
    {

    strcat(cur-label->attr-pnt->pnt-info,"~");              /* concatinate unary '-' onto string *
**
    strcat(cur-label->attr-pnt->pnt-info,":");              /* terminate */
    unary= NO;

    }
```

```
/*+
** NAME:
**              error
**
** SYNOPSIS:
**              int error(error_num)
**                      int     error_num;
**
** PURPOSE:
**              Error routine for the gpss parser
**
** RETURNS:
**              ERROR   - error signifying syntatic error
**
** MAINTENANCE:
**              27-June-83      jvd     created
**
**
-*/
#include        stdio

#include        "[vandellon.thesis]gpss"

globalref       char    line[81];
globalref       int     line_number;
globalref       int     block_number;
globalref       int     end_line_pointer;
globalref       int     error_count;

int error(error_num)
    int error_num;
{
    int         i;
    char        temp_line[81];

    for(i= 0 ; i < (end_line_pointer + 10) ; i++)
        temp_line[i]= ' ';

    temp_line[(end_line_pointer + 10)]= '*';
    temp_line[(end_line_pointer + 11)]= NULL;

    switch(error_num)
        {
        case(NO_END):
                printf("\n>> No END statement encountered\n");
                break;
        case(REQ_OPER):
                printf("          %s\n",temp_line);
                printf("                >> Required operands missing\n");
                break;
        case(LA8_NOT_ALW):
                printf("          %s\n",temp_line);
                printf("                >> Label not allowed\n");
                break;
        case(NO_OPER):
                printf("          %s\n",temp_line);
                printf("                >> Operand is not valid\n");
                break;
        case(UN_OPER):
                printf("          %s\n",temp_line);
                printf("                >> Undefined operand\n");
                break;
        case(SYNTAX):
                printf("          %s\n",temp_line);
                printf("                >> Syntax error\n");
                break;
        case(ILL_OPT):
                printf("          %s\n",temp_line);
                printf("                >> Illegal option specified\n");
                break;
        case(ILL_LAB):
                printf("          %s\n",temp_line);
                printf("                >> Illegal specification of label\n");
                break;
        case(OPEN_ERR):
                printf(">> Open error on input file\n");
```

```
            break;
        case(REP_LABEL):
                printf("          %s\n",temp_line);
                printf("               >> Multiply defined label\n");
                break;
        case(MULT_SIM):
                printf("          %s\n",temp_line);
                printf("               >> Multiple SIMULATION statements\n");
                break;
        case(MULT_END):
                printf("          %s\n",temp_line);
                printf("               >> Multiple END statements\n");
                break;
        case(TOO_START):
                printf("          %s\n",temp_line);
                printf("               >> 50 START statements allowed\n");
                break;
        case(LAB_REQ):
                printf("          %s\n",temp_line);
                printf("               >> Label required\n");
                break;
        case(ILL_NONNUM):
                printf("          %s\n",temp_line);
                printf("               >> Illegal specification of numeric quantity\n");
                break;
        case(ILL_OPER):
                printf("          %s\n",temp_line);
                printf("               >> Illegal specification of operand\n");
                break;
        case(LAB_NO_STAT):
                printf("          %s\n",temp_line);
                printf("               >> Label valid but no statement\n");
                break;
        case(PRE_END):
                printf("          %s\n",temp_line);
                printf("               >> Premature end of file\n");
                break;
        case(ILL_FUNC):
                printf("          %s\n",temp_line);
                printf("               >> Illegal function definition\n");
                break;
        case(MIS_FUNC):
                printf("          %s\n",temp_line);
                printf("               >> Missing function definition\n");
                break;
        case(OP_NVLD):
                printf("          %s\n",temp_line);
                printf("               >> Operand not valid in this version\n");
                break;
        case(ILL_VAR_DEF):
                printf("          %s\n",temp_line);
                printf("               >> Illegal variable definition\n");
                break;
        case(PAR10):
                printf("          %s\n",temp_line);
                printf("               >> A maximum of 10 parameters are allowed\n");
                break;
        case(COMPILER):
                printf("          %s\n",temp_line);
                printf("               >> COMPILER error\n");
                break;
        case(UN_LABEL):
                printf(">>> Undefined label %s\n",line);
                break;
        case(STR_NOT_FND):
                printf(">>> SIMULATOR ERROR <<< Storage out of range or undefined\n");
                break;
        case(FINISH_ERR):
                printf("\nTotal number of errors - %d",error_count);
                printf("\nSimulation phase will not occur due to ERRORS\n");
                printf("*       GPSS exiting    *\n");
                break;
    }
    error_count++;
    return(ERROR);
}
```

```
/*+
** NAME:
**              getvar
**
** SYNOPSIS:
**              char *getvar()
**
** PURPOSE:
**              Get a word from a variable line.
**
** RETURNS:
**              WORD    - char pointer to the gotten word
**
** MAINTENANCE:
**              9-June-83       jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       char    line[];
globalref       int     line+pointer;
globalref       int     end+line+pointer;

char *getvar()
{
    static char word[80]= 0;
    int         i;

    if( end+line+pointer == 79 )                                /* check for end of line */
        return(EOL);

    while((isspace(line[end+line+pointer])) && (end+line+pointer != 79))/* skip over white space to nex
**t word */
        end+line+pointer++;

    line+pointer= end+line+pointer;                             /* make beginning & end pointer equival
**ent */
    if(line[end+line+pointer] == NULL)                          /* check for end of line by line feed *
**/
        return(EOL);
    else if((line[end+line+pointer] == '+') || (line[end+line+pointer] == '-')
                || (line[end+line+pointer] == '*') || (line[end+line+pointer] == '/'))/* check for arit
**hmetic operations */
        {
        end+line+pointer++;
        return(line[end+line+pointer - 1]);
        }
    else if(line[end+line+pointer] == '!')                      /* check for comment */
        return(EOL);

    if((line[end+line+pointer] == 'v') || (line[end+line+pointer] == 'V') ||
        (line[end+line+pointer] == 'p') || (line[end+line+pointer] == 'P') ||
        (line[end+line+pointer] == 'f') || (line[end+line+pointer] == 'F'))
        {
        if((line[end+line+pointer + 1] == '*') || (line[end+line+pointer + 1] == 'n'))/* want to make s
**ure correct specification */
                ;
        else
                return(error(SYNTAX));

        for(i= 0 ;
                (isalnum(line[end+line+pointer])) || (line[end+line+pointer] == '*') && (end+line+point
**er < 79) ;
                end+line+pointer++, i++)                        /* look char by char for complete word
***/
                word[i]= line[end+line+pointer];
        }
    else if(isdigit(line[end+line+pointer]))
        {
        for(i= 0 ;
                (isdigit(line[end+line+pointer])) && (end+line+pointer < 79) ;
                end+line+pointer++, i++)                        /* look char by char for complete word
```

```
***/
                word[i]= line[end-line-pointer];
        }
    else
        return(error(SYNTAX));

    word[i]= NULL;                                          /* null terminate word */
    return(word);
}
```

```
/*+
** NAME:
**              getword
**
** SYNOPSIS:
**              char *getword()
**
** PURPOSE:
**              Get a word from the source line.
**
** RETURNS:
**              WORD    - char pointer to the gotten word
**
** MAINTENANCE:
**              9-June-83       jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       char    line[81];
globalref       int     line_pointer;
globalref       int     end_line_pointer;

char *getword()
{
    static char word[80]= 0;
    int         i;

    if( end_line_pointer == 79 )                                /* check for end of line */
        return(EOL);

    while((isspace(line[end_line_pointer])) && (end_line_pointer != 79))/* skip over white space to nex
**t word */
        end_line_pointer++;

    line_pointer= end_line_pointer;                             /* make beginning & end pointer equival
**ent */
    if(line[end_line_pointer] == NULL)                          /* check for end of line by line feed *
**/
        return(EOL);
    else if(line[end_line_pointer] == ',')                      /* check for COMMA, special character *
**/
        {
        end_line_pointer++;
        return(COMMA);
        }
    else if(line[end_line_pointer] == '!')                      /* check for comment */
        return(EOL);

    for(i= 0 ;
        (isalnum(line[end_line_pointer])) || (line[end_line_pointer] == '_') || (line[end_line_pointer]
** == '*') && (end_line_pointer < 79) ;
        end_line_pointer++, i++)                                /* look char by char for complete word
***/
        word[i]= line[end_line_pointer];


    word[i]= NULL;                                              /* null terminate word */
    return(word);
}
```

```
/*+
** NAME:
**              itoa
**
** SYNOPSIS:
**              char *itoa()
**
** PURPOSE:
**              Change integer to ascii, specifically the temporary variable
**              number.
**
** RETURNS:
**              returns converted string
**
** MAINTENANCE:
**              12-feb-84       jvd     created
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"

globalref       int     temp_var;

char *itoa()
{
    int         hold;
    int         i;
    int         work_number;
    static char conv_num[4];
    int         divisor= 100;

    conv_num[3]= NULL;
    work_number= temp_var;

    for(i= 0 ; i < 3 ; i++)
        {
        hold= work_number / divisor;
        if((hold >= 0) && (hold <= 9))
                conv_num[i]= (hold + 48);               /* change the number into ascii format */
        work_number= work_number - (hold * divisor);
        divisor= divisor / 10;
        }

    temp_var++;                                         /* increase the temporary variable count */
    return(conv_num);
    }
```

```
/*+
** NAME:
**              keyword
**
** SYNOPSIS:
**              int keyword(string)
**                      char    *string
**
** PURPOSE:
**              Compare character string to see if keyword
**
** RETURNS:
**              Integer value for keyword if it is or negative if not
**
** MAINTENANCE:
**              14-June-83      jvd     created
**
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"

int keyword(string)
    char        *string;
{

    if((strcmp(string,"SIMULATE") == 0) || (strcmp(string,"simulate") == 0))
                return(SIMULATE);
    else if((strcmp(string,"START") == 0) || (strcmp(string,"start") == 0))
        return(START);
    else if((strcmp(string,"END") == 0) || (strcmp(string,"end") == 0))
        return(END);
    else if((strcmp(string,"STORAGE") == 0) || (strcmp(string,"storage") == 0))
        return(STORAGE);
    else if((strcmp(string,"VARIABLE") == 0) || (strcmp(string,"variable") == 0))
        return(VARIABLE);
    else if((strcmp(string,"FUNCTION") == 0) || (strcmp(string,"function") == 0))
        return(FUNCTION);
    else if((strcmp(string,"GENERATE") == 0) || (strcmp(string,"generate") == 0))
        return(GENERATE);
    else if((strcmp(string,"TERMINATE") == 0) || (strcmp(string,"terminate") == 0))
        return(TERMINATE);
    else if((strcmp(string,"ADVANCE") == 0) || (strcmp(string,"advance") == 0))
        return(ADVANCE);
    else if((strcmp(string,"ASSIGN") == 0) || (strcmp(string,"assign") == 0))
        return(ASSIGN);
    else if((strcmp(string,"SEIZE") == 0) || (strcmp(string,"seize") == 0))
        return(SEIZE);
    else if((strcmp(string,"RELEASE") == 0) || (strcmp(string,"release") == 0))
        return(RELEASE);
    else if((strcmp(string,"ENTER") == 0) || (strcmp(string,"enter") == 0))
        return(ENTER);
    else if((strcmp(string,"LEAVE") == 0) || (strcmp(string,"leave") == 0))
        return(LEAVE);
    else if((strcmp(string,"QUEUE") == 0) || (strcmp(string,"queue") == 0))
        return(QUEUE);
    else if((strcmp(string,"DEPART") == 0) || (strcmp(string,"depart") == 0))
        return(DEPART);
    if((strcmp(string,"GATE") == 0) || (strcmp(string,"gate") == 0))
        return(GATE);
    else if((strcmp(string,"TEST") == 0) || (strcmp(string,"test") == 0))
        return(TEST);
    else if((strcmp(string,"TRANSFER") == 0) || (strcmp(string,"transfer") == 0))
        return(TRANSFER);
    else
        return(-1);
}
```

```
/*+
** NAME:
**              label
**
** SYNOPSIS:
**              int label(newword,root)
**                      char    *newword;
**                      SYM-TABLE *root;
**
** PURPOSE:
**              Process label.
**
** RETURNS:
**              ERROR   - if multiply defined label
**              SUCCESS - OK
**
** MAINTENANCE:
**              12-July-3       jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       char    line[];
globalref       int     line-pointer;
globalref       int     line-number;
globalref       int     label-light;
globalref       SYM-TABLE *cur-label;

int label(newword,root)
    char        *newword;
    SYM-TABLE   *root;
    {
    int ret-status;
    int i;

    if((*newword == NULL) || (label-light == ON))/* check for illegal label signified by a null */
                                                 /* or a label already on this line */
        {
        printf("%d              %s",line-number,line);
        return(error(SYNTAX));                   /* syntax error printed */
        }
    else
        {
        for(i= line-pointer ; isalpha('ine[i]) ; i++)
                ;
        for( ; (isspace(line[i]) != 0) && (line[i] != NULL) ;   /* move cursor across line looking for
**more on line than label */
                i++)
                ;
        if((line[i] == NULL) || (line[i] == '!'))
                {
                printf("%d              %s",line-number,line);
                return(error(LAB-NO-STAT));
                }
        }

    if((ret-status= strlab(root,newword,0)) == REP-LABEL)
        {
        printf("%d              %s",line-number,line);
        return(error(ret-status));
        }

    cur-label->attr-pnt->lab-attribute= LAB-LABEL;

    return(SUCCESS);            /* if AOK store label into the symbol table */
    }
```

```
/*+
** NAME:
**              prsflo
**
** SYNOPSIS:
**              int prsflo(j)
**                      int j;
**
** PURPOSE:
**              Parse a word, examining the word to make sure it is a floating
**              point number.
**
** RETURNS:
**              ERROR     the number found was not floating point
**              float     a floating point number
**
** MAINTENANCE:
**              21-Jan-84       jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       char    line[];
globalref       int     end-line-pointer;
globalref       int     line-pointer;
globalref       FILE    *fileptr;
globalref       SYM-TABLE *cur-label;
globalref       int     line-number;

int prsflo(j)
    int         j;              /* place to put information */
    {
    char        *word;
    int         i;
    char        *buf;
    double      atof();

    if(line[end-line-pointer] == '0')                           /* check for preceding zero */
        end-line-pointer++;

    if(line[end-line-pointer] == '.')                           /* make sure decimal there */
        {
        end-line-pointer++;
        if(isdigit(line[end-line-pointer]))                     /* make sure numeric follows decimal */
**
                {
                if(((word= getword()) == EOL) || (word == COMMA))
                        return(error(SYNTAX));
                if(prsnum(word,FUNCTION) == ERROR)
                        return(ERROR);
                else
                        {
                        buf= malloc(sizeof(word) + 2);          /* put number in form for atof conversi
**on. */
                        *buf= '.';
                        strcat(buf,word);
                        buf[sizeof(word) + 2]= NULL;
                        *((cur-label->attr-pnt->flo-num) + j)= atof(buf);/* put info into symbol table
***/
                        }
                }
        else
                return(ERROR);
        }
    else if(line[end-line-pointer] == 012)
        {
        if(fgets(line,MAXLINE,fileptr) == NULL)
                return(error(PRE-END));
        else
                {
                line-number++;
                end-line-pointer= line-pointer= 0;
```

```
                printf("%d\t    %s",line_number,line);
                }
        if(prsflo(j) == ERROR)
                return(ERROR);
        }
else
        return(error(SYNTAX));

return(SUCCESS);
}
```

```
/*+
** NAME:
**              prslab
**
** SYNOPSIS:
**              prslab(hold←label)
**                      SYM←TABLE       *hold←label;
**
** PURPOSE:
**              Parse for alphanumeric label.  And if label, store for use later.
**
** RETURNS:
**              ERROR   - signifies error in label
**              SUCCESS - OK
**
** MAINTENANCE:
**              8-jan-83        jvd     created
**              18-Mar-84       jvd     add storage of cur←label
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       SYM←TABLE       *cur←label;
globalref       int             label←light;
globalref       CODE            *ps←cur;

int prslab(hold←label)
    SYM←TABLE   *hold←label;
    {

    if(label←light == ON)                                       /* check to make sure labellight is on
***/
        {
        cur←label->attr←pnt->lab←attribute= LAB←LABEL;          /* else mark it as a label */
        cur←label->attr←pnt->pnt←info= ps←cur;                  /* point label to pseudo code statement
** */
        if(isdigit(cur←label->label[0]))                        /* if digit first thing ERROR */
                return(error(ILL←LAB));
        hold←label= cur←label;
        }

    return(SUCCESS);
    }
```

```
/*+
** NAME:
**              prsnum
**
** SYNOPSIS:
**              int prsnum(word,root)
**                      char    *word;
**                      SYM←TABLE *root;
** PURPOSE:
**              Parse a word, examining the word to make sure it is a number.
**              If correct number put into symbol table for use in pseudo code
**              generation.
**
** RETURNS:
**              ERROR   - error if not a valid number
**              SUCCESS - if good number and created entry in symbol table
**
** MAINTENANCE:
**              15-Nov-83       jvd     created
**              18-MAR-84       jvd     addition of pseudo code generation
**                                      of CONSTANT
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       char    temp←label[];
globalref       SYM←TABLE *cur←label;

int prsnum(word,root)
    char        *word;
    SYM←TABLE   *root;
    {
    int         i;

    for(i= 0 ; isdigit(*(word + i)) && (i <= strlen(word)) ; i++)/* check make sure all chars are digit
**s */
        ;

    if( (i == strlen(word)) && (i != 0) )                       /* check to make sure OK */
        {
        if((root == FUNCTION) || (root == STORAGE) || (root == VARIABLE))/* check for constants on bloc
**k statements */
                return(SUCCESS);
        temp←label[3]= NULL;
        if(strlab(root,strcat(temp←label,itoa()),0) == ERROR)   /* enter the number as a constant in th
**e symbol table */
                return(error(SYNTAX));
        cur←label->attr←pnt->lab←attribute= CONSTANT;           /* label new variable as CONSTANT */
        cur←label->attr←pnt->leng←info= 0;                      /* just have an integer */
        cur←label->attr←pnt->pnt←info= malloc(sizeof(int));     /* get room for integer information */
        *cur←label->attr←pnt->pnt←info= atoi(word);             /* put inforamtion in symbol table */
        cur←label->attr←pnt->func←spec= 0;                      /* zero out inapplicable stuff */
        cur←label->attr←pnt->flo←num= 0;
        return(SUCCESS);
        }
    else
        return(error(ILL←NONNUM));

    }
```

```
/*+
** NAME:
**              prsolb  - parse a word, examining for a parameter, storage,
**                        function, or variable definition such as p*1 or P*1
**
** SYNOPSIS:
**              int prsolb(word)
**                      char    *word;
** PURPOSE:
**              Parse a word, examining for a parameter, storage, function, or
**              variable definiton such as p*1 or P*1.
**
** RETURNS:
**              ERROR   - signifying an invalid SNA
**              SNA←RANOOM      a random number
**              SNA←PARAM       a parameter SNA
**              LAB←VAR         a variable SNA
**              LAB←STOR        a storage SNA
**              LAB←FUNC        a function SNA
**
** MAINTENANCE:
**              12-Oec-B3       jvd     created
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       SYM←TABLE       *cur←label;

int prsolb(word,root)
    char        *word;
    SYM←TABLE   *root;
    {
    char        temp[2];
    SYM←TABLE   *ret←root;

    temp[0]= *word;
    temp[1]= NULL;

    if(strcspn(temp,"psfvrPSFVR") == strlen(temp))
        return(ERROR);

    if((temp[0] == 'f') || (temp[0] == 'F'))
        {
        if((*(word + 1) == 'N') || (*(word + 1) == 'n'))
                ++word;
        else
                return(ERROR);
        }
    ++word;

    if(((temp[0] == 'r') || (temp[0] == 'R')) && (cur←label->attr←pnt->lab←attribute != LAB←FUNC))
        return(ERROR);

    if(*word != '*')
        return(ERROR);
    ++word;

    if((isdigit(*word) && (temp[0] == 'p')) || (isdigit(*word) && (temp[0] == 'P')))
        {
        if(prsnum(word,root) == ERROR)
                return(ERROR);
        else
                {
                cur←label->attr←pnt->lab←attribute= SNA←PARAM;
                if(atoi(word) > 10)
                        return(error(PAR10));
                return(SNA←PARAM);
                }
        }
    else if((temp[0] == 's') || (temp[0] == 'S'))
        {
        if(isdigit(*word))
                {
                if(prsnum(word,STORAGE) == ERROR)
                        return(ERROR);
```

```
                }
        if((ret←root= strlab(root,word,SEARCH)) == ERROR)
                {
                error(UN←OPER);
                return(ERROR);
                }
        else if(ret←root->attr←pnt->lab←attribute != LAB←STOR)
                return(ERROR);
        else
                {
                cur←label= ret←root;                            /* pass info back to routine */
                return(LAB←STOR);
                }
        }
    else if((temp[0] != 'p') && (temp[0] != 'P'))
        {
        if((ret←root= strlab(root,word,SEARCH)) == ERROR)
                {
                error(UN←OPER);
                return(ERROR);
                }
        if(ret←root->attr←pnt->lab←attribute == SNA←PARAM)
                return(ERROR);
        if(((temp[0] == 'r') || (temp[0] == 'R')) && (ret←root->attr←pnt->lab←attribute == SNA←RANOOM))
**
                return(SNA←RANDOM);
        else if(((temp[0] == 'v') || (temp[0] == 'V')) && (ret←root->attr←pnt->lab←attribute == LAB←VAR
**))
                {
                cur←label= ret←root;                            /* pass info back to routine */
                return(LAB←VAR);
                }
        else if(((temp[0] == 'f') || (temp[0] == 'F')) && (ret←root->attr←pnt->lab←attribute == LAB←FUN
**C))
                {
                cur←label= ret←root;                            /* pass info back to routine */
                return(LAB←FUNC);
                }
        else
                return(ERROR);
        }
    else
        return(ERROR);

    }
```

```
/*+
** NAME:
**              strlab
**
** SYNOPSIS:
**              SYM-TABLE *strlab(root,text-label,key)
**                  SYM-TABLE   *root;
**                  char        *text-label;
**                  int         key;
**
** PURPOSE:
**              Store the label in the symbol table.  The label can be storage,
**              a variable, a function, or a lowly label.
**
** RETURNS:
**              REP-LABEL       - duplicate label
**              error           - could not find label
**              root            - found label and return pointer
**
** MAINTENANCE:
**              31-Sept-83      jvd     created
**
-*/
#include        stdio
#include        descrip
#include        ssdef
#include        "[vandellon.thesis]gpss"

globalref       int     end-line-pointer;
globalref       int     line-pointer;
globalref       SYM-TABLE *cur-label;

SYM-TABLE *strlab(root,text-label,key)
    SYM-TABLE   *root;
    char        *text-label;
    int         key;
    {
    SYM-TABLE   *ret-status;

    if((root->label == NULL) && (key == SEARCH))                /* can't find label in table */
        return(ERROR);
    else if((root->label == NULL) && (key != SEARCH))           /* found place to put new label */
        {
        cur-label= root;                                        /* create structure for new label */
        root->left= malloc(sizeof(SYM-TABLE));
        root->right= malloc(sizeof(SYM-TABLE));
        root->label= malloc((end-line-pointer - line-pointer) + 2);
        strcpy(root->label,text-label);
        root->attr-pnt= malloc(sizeof(LAB));
        }
    else if(((ret-status= strcmp(text-label,root->label)) == 0) && (key != SEARCH))/* make sure not a r
**epeat label */
        return(REP-LABEL);                                      /* if so error */
    else if((ret-status == 0) && (key == SEARCH))               /* found label */
        return(root);
    else if(ret-status < 0)                                     /* recursive search through tree */
        return(strlab((root->left),text-label,key));
    else
        return(strlab((root->right),text-label,key));

     }
```

```
/*+
** NAME:
**              vaddsub
**
** SYNOPSIS:
**              int vaddsub(word,root,hold←label)
**                      char            *word
**                      SYM←TABLE       *root;
**                      SYM←TABLE       *hold←label;
**
** PURPOSE:
**              Parse variables with '+' or '-'
**
** RETURNS:
**              ERROR   - syntactic error detected
**
** MAINTENANCE:
**              19-Feb-84       jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       SYM←TABLE *cur←label;
globalref       int     unary;

int vaddsub(word,root,hold←label)
    SYM←TABLE   *root;
    char        *word;
    SYM←TABLE   *hold←label;
    {
    char        operator[2];
    char        *strip←pnt;

    operator[0]= word;
    operator[1]= NULL;

    if(((word= getvar()) == EOL) || (word == COMMA))
        return(error(NO←OPER));
    else if(word == ERROR)
        return(ERROR);
    else if((word == ADDITION) || (word == MULTIPLICATION) ||
        (word == DIVISION))
        return(error(SYNTAX));

    if(word  == SUBTRACTION)
        {
        unary= YES;
        if(((word= getvar()) == EOL) || (word == COMMA))
                return(error(NO←OPER));
        else if(word == ERROR)
                return(ERROR);
        else if((word == ADDITION) || (word == SUBTRACTION) ||
                (word == MULTIPLICATION) || (word == DIVISION))
                return(error(SYNTAX));
        }

    if(isdigit(*word))
        {
        if(prsnum(word,VARIABLE) == ERROR)
                return(ERROR);
        }
    else if(prsolb(word,root) != ERROR)
        {
        strip←pnt= strpbrk(word,"*");
        strcpy(word,(strip←pnt + 1));
        cur←label= hold←label;
        }
    else
        return(ERROR);

    strcat(cur←label->attr←pnt->pnt←info,word);
    strcat(cur←label->attr←pnt->pnt←info,":");
```

```
if(unary == YES)
    conunary();

strcat(cur->label->attr->pnt->pnt->info.operator);
strcat(cur->label->attr->pnt->pnt->info.":");

}
```

```
/*+
** NAME:
**              vmuldiv
**
** SYNOPSIS:
**              int vmuldiv(word,root)
**                      char            *word;
**                      SYM-TABLE       *root;
**
** PURPOSE:
**              Parse multiplication and division for VARIABLE statement.
**
** RETURNS:
**              ERROR   - syntactic error detected
**
** MAINTENANCE:
**              19-Feb-84       jvd     created
**
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"

globalref       SYM-TABLE *cur-label;
globalref       int     unary;
globaldef       char    temp-label[7]= { 'v','1','t', NULL };                   /* temporary variable n
**ame */

int vmuldiv(word,root)
    char        *word;
    SYM-TABLE   *root;
    {
    char        sav-oper1[2];
    char        sav-oper2[2];
    char        hold-operand[20];
    char        *right;
    SYM-TABLE   *hold-var;
    char        *strrchr();

    sav-oper1[1]= sav-oper2[1]= NULL;

    right= strrchr(cur-label->attr-pnt->pnt-info,':');          /* find right most terminator */
    *right= NULL;                                               /* erase it */

    right= strrchr(cur-label->attr-pnt->pnt-info,':');          /* find next right most terminator */

    if(right != 0)
        {
        if(*(right + 1) == UNARY)
                unary= YES;
        else
                sav-oper1[0]= *(right + 1);                     /* save operator */

        *right= NULL;                                           /* clear flag */
        right= strrchr(cur-label->attr-pnt->pnt-info,':');      /* get position for operand */

        if(right == 0)                                          /* found operand */
                {
                strcpy(hold-operand,cur-label->attr-pnt->pnt-info);
                *cur-label->attr-pnt->pnt-info= NULL;
                }
        else
                {
                if(*(right + 1) == UNARY)
                        {
                        unary= YES;
                        *right= NULL;
                        right= strrchr(cur-label->attr-pnt->pnt-info,':');/* get position for operand *
**/

                        if(right == 0)                                  /* found operand */
                                {
                                strcpy(hold-operand,cur-label->attr-pnt->pnt
                                *cur-label->attr-pnt->pnt-info= NULL;
```

```
                        }
                    else
                        {
                        strcpy(hold->operand,(right + 1));
                        *(right + 1)= NULL;
                        }
                    }
                else
                    {
                    strcpy(hold->operand,(right + 1));
                    *(right + 1)= NULL;
                    }
                }
            }
        else                                                    /* if no right most terminator */
            {
            strcpy(hold->operand,cur->label->attr->pnt->pnt->info);     /* copy operand into holder */
            *cur->label->attr->pnt->pnt->info= NULL;                    /* clear current label info */
            }

        temp->label[3]= NULL;
        hold->var= cur->label;                                  /* save current variable 'cause could m
**ake temp->varaible */

        if(strlab(root,strcat(temp->label,itoa()),0) == ERROR)      /* enter temp->variable into the symbol
**table */
            return(error(SYNTAX));
        cur->label->attr->pnt->lab->attribute= TEMP->VAR;           /* give it status of temporary */
        cur->label->attr->pnt->pnt->info= malloc(sizeof(int) * 80); /* allocate space for info */
        *cur->label->attr->pnt->pnt->info= NULL;

        strcat(cur->label->attr->pnt->pnt->info,hold->operand);     /* put operand into new variable */
        strcat(cur->label->attr->pnt->pnt->info,":");               /* terminate */
        if(unary == YES)
            conunary();

        sav->oper2[0]= word;                                    /* save '*' or '/' for later use */

        if(((word= getvar()) == ERROR) || (word == ADDITION) ||
            (word == MULTIPLICATION) || (word == DIVISION))
            return(error(ILL->VAR->DEF));

        if(word == SUBTRACTION)
            {
            unary= YES;
            if((word= getvar()) == EOL)
                    return(error(NO->OPER));
            else if(word == ERROR)
                    return(ERROR);
            else if((word == ADDITION) || (word == SUBTRACTION) ||
                    (word == MULTIPLICATION) || (word == DIVISION))
                    return(error(ILL->VAR->DEF));
            }

        strcat(cur->label->attr->pnt->pnt->info,word);
        strcat(cur->label->attr->pnt->pnt->info,":");
        if(unary == YES)
            conunary();

        strcat(cur->label->attr->pnt->pnt->info,sav->oper2);
        strcat(cur->label->attr->pnt->pnt->info,":");
        strcat(hold->var->attr->pnt->pnt->info,cur->label->label);
        strcat(hold->var->attr->pnt->pnt->info,":");
        if((right != 0) || ((right == 0) && (unary == YES)))
            {
            strcat(hold->var->attr->pnt->pnt->info,sav->oper1);
            strcat(hold->var->attr->pnt->pnt->info,":");
            }
        cur->label->attr->pnt->leng->info= strlen(cur->label->attr->pnt->pnt->info);
        cur->label= hold->var;

        }
```

```
/*+
** NAME:
**              prsymtab
**
** SYNOPSIS:
**              prsymtab(root)
**                      SYM←TABLE       *root;
**
** PURPOSE:
**              Printing symbol table values.  ***TEST*** purposes only.
**
** RETURNS:
**              Not applicable.
**
** MAINTENANCE:
**              29-Dec-83       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"

globaldef int           count= 0;
globalref int           strt←ptr;
globalref INFO←STRT     *strt[];

int prsymtab(root)
    SYM←TABLE   *root;
    {
    char        hold[10];
    int         i;

    if(root->label == NULL)
        return;

    if(count == 0)
        {
        printf("\n\n*************************************************\n");
        printf("\nSymbol Table Contents after First Pass Completion");
        printf("\n=================================================");
        count++;
        }

    printf("\n\nLabel: %s",root->label);
    if((root->attr←pnt->lab←attribute) == LAB←STOR)
        strcpy(hold,"STORAGE");
    else if((root->attr←pnt->lab←attribute) == LAB←LABEL)
        strcpy(hold,"LABEL");
    else if((root->attr←pnt->lab←attribute) == LAB←VAR)
        strcpy(hold,"VARIABLE");
    else if((root->attr←pnt->lab←attribute) == TEMP←VAR)
        strcpy(hold,"TEMPORARY");
    else if((root->attr←pnt->lab←attribute) == LAB←FUNC)
        strcpy(hold,"FUNCTION");
    else if((root->attr←pnt->lab←attribute) == CONSTANT)
        strcpy(hold,"CONSTANT");
    else if((root->attr←pnt->lab←attribute) == SNA←PARAM)
        strcpy(hold,"PARAMETER");
    else
        strcpy(hold,"********");

    hold[strlen(hold)]= NULL;
    printf("\n\tLabel attribute: %s",hold);
    if(root->attr←pnt->leng←info != 0)
        {
        if(root->attr←pnt->lab←attribute == LAB←STOR)
                {
                printf("\n\tNumber of storage units: %d\n",root->attr←pnt->leng←info);
                printf("\tNumber of units left in storage: %d\n",root->attr←pnt->func←spec);
                }
        if(root->attr←pnt->lab←attribute == LAB←FUNC)
                {
                printf("\n\tNumber of points in the function: %d",root->attr←pnt->func←spec);
                for(i= 0 ; i < root->attr←pnt->leng←info ; i++)
                        printf("\n\t\t%lf       %d",*(root->attr←pnt->flo←n
                                                    *(root->attr←pnt->pnt←info
```

```
            }
        printf("\n\tLength of information: %d",root->attr->pnt->leng->info);
        if((root->attr->pnt->lab->attribute == LAB->VAR) || (root->attr->pnt->lab->attribute == TEMP->VAR))
                {
                printf("\n\tInformation stored:");
                printf("\n\t\t%s",root->attr->pnt->pnt->info);
                }
        }
    else if((root->attr->pnt->lab->attribute == CONSTANT) ||
                (root->attr->pnt->lab->attribute == SNA->PARAM))
        {
        printf("\n\tInformation stored:");
        printf("\n\t\t%d",*(root->attr->pnt->pnt->info));
        }

    if(root->left != NULL)
        prsymtab(root->left);

    if(root->right != NULL)
        prsymtab(root->right);

    if(count == 1)
        {
        printf("\n*****************************************************\n\n");
        printf("======================\n");
        printf("START card information\n");
        printf("======================\n");

        for(i= 0 ; strt->ptr > i ; i++)
                {
                printf("\nSTART card #%d:\tsimulation time %d\n",(i+1),strt[i]->sim->time);
                if(strt[i]->print == PRT)
                        {
                        printf("\t\tSNAP interval printout requested\n");
                        printf("\t\tInterval printout time is %d units\n",strt[i]->interval->print);
                        }
                else
                        printf("\t\tNO SNAP interval printout requested\n");
                }
        count++;
        }

    }
```

12.2. Pass 2 Program Listings.

*** NOTE ***
The following pages are the program listings for the thesis project. Due to printing capabilities of the hardware, an underscore is an undefined character. The ← character is the representation for the underscore.

```
/*+
** NAME:
**              gps-pass2
**
** SYNOPSIS:
**              int gps-pass2()
**
** PURPOSE:
**              Main routine for pass2 of the compiler.  All pass2 does is
**              complete the creation of the pseudo code by filling in labels.
**
** RETURNS:
**              0       - if no errors
**              -1      - if errors
**
** MAINTENANCE:
**              24-Mar-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"

globalref       char    line[];
globalref       CODE    *ps-cur;
globalref       CODE    *ps-root;

int gps-pass2(root)
    SYM-TABLE   *root;
{
    CODE        *ps-work;
    int         ret-status;

    ps-work= ps-root;

    for( ; ps-work != ps-cur ; ps-work= ps-work->code-link)
        {
        if((ps-work->operator == GT) || (ps-work->operator == EQ) ||
           (ps-work->operator == LT) || (ps-work->operator == U) ||
           (ps-work->operator == NU) || (ps-work->operator == SF) ||
           (ps-work->operator == SE) || (ps-work->operator == SNF) ||
           (ps-work->operator == SNE))
                {
                ps-work= ps-work->code-link;
                if( ps-work == ps-cur)
                        break;
                else if((int)ps-work->A-operand->attr-pnt == LAB-LABEL)
                        {
                        if((ret-status= strlab(root,ps-work->A-operand->label,SEARCH)) == ERROR)
                                {
                                strcpy(line,ps-work->A-operand->label);
                                error(UN-LABEL);
                                }
                        else
                                ps-work->A-operand= ret-status;
                        }
                }
        else if(ps-work->operator == BR)
                {
                if((int)ps-work->B-operand->attr-pnt == LAB-LABEL)
                        {
                        if((ret-status= strlab(root,ps-work->B-operand->label,SEARCH)) == ERROR)
                                {
                                strcpy(line,ps-work->B-operand->label);
                                error(UN-LABEL);
                                }
                        else
                                ps-work->B-operand= ret-status;
                        }
                }
        }
    return;
}
```

```
/*+
** NAME:
**              crepsd
**
** SYNOPSIS:
**              int crepsd(operator)
**                      int     operator;
**
** PURPOSE:
**              Create the pseudo code structure for GPSS code translation.
**
** RETURNS:
**              No returning value.
**
** MAINTENANCE:
**              18-Mar-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"

globalref       COOE    *ps←cur;

int crepsd(operator)
    int operator;
    {

    ps←cur->operator= operator;
    ps←cur->code←link= malloc(sizeof(COOE));

    }
```

```
/*+
** NAME:
**              ps2tst
**
** SYNOPSIS:
**              ps2tst()
**
** PURPOSE:
**              Phase two test print out of pseudo code generated.
**              **** TEST **** purposes only.
**
** RETURNS:
**              Not applicable.
**
** MAINTENANCE:
**              21-Mar-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"

globalref       CODE    *ps-root;
globalref       CODE    *ps-cur;

int ps2tst()
    {
    CODE        *ps-work;

    printf("\n\n\n====================");
    printf("\nPseudo Code Generated\n");
    printf("====================\n\n");

    ps-work= ps-root;
    for(; ps-work != ps-cur ; ps-work= ps-work->code-link)
        {
        switch(ps-work->operator)
                {
                case(GENERATE):
                        printf("\noperator: GENERATE\n");
                        break;

                case(ASSIGN):
                        printf("\noperator: ASSIGN\n");
                        break;

                case(ADVANCE):
                        printf("\nopera*or: ADVANCE\n");
                        break;

                case(DEPART):
                        printf("\noperator: DEPART\n");
                        break;

                case(ENTER):
                        printf("\noperator: ENTER\n");
                        break;

                case(LEAVE):
                        printf("\noperator: LEAVE\n");
                        break;

                case(SEIZE):
                        printf("\noperator: SEIZE\n");
                        break;

                case(QUEUE):
                        printf("\noperator: QUEUE\n");
                        break;

                case(RELEASE):
                        printf("\noperator: RELEASE\n");
                        break;

                case(BR):
                        printf("\noperator: BR\n");
```

```
                        break;

                case(GATE):
                        printf("\noperator: GATE\n");
                        break;

                case(TEST):
                        printf("\noperator: TEST\n");
                        break;

                case(TERMINATE):
                        printf("\noperator: TERMINATE\n");
                        break;

                case(BNE):
                        printf("\noperator: BNE\n");
                        break;

                case(BE):
                        printf("\noperator: BE\n");
                        break;

                case(EQ):
                        printf("\noperator: EQ\n");
                        break;

                case(GT):
                        printf("\noperator: GT\n");
                        break;

                case(LT):
                        printf("\noperator: LT\n");
                        break;

                case(U):
                        printf("\noperator: U\n");
                        break;

                case(NU):
                        printf("\noperator: NU\n");
                        break;

                case(SF):
                        printf("\noperator: SF\n");
                        break;

                case(SE):
                        printf("\noperator: SE\n");
                        break;

                case(SNF):
                        printf("\noperator: SNF\n");
                        break;

                case(SNE):
                        printf("\noperator: SNE\n");
                        break;

                case(NEG):
                        printf("\noperator: NEG\n");
                        break;
                }
        printf("block number = %d\n",ps->work->block);
        if(ps->work->A->operand != NULL)
                printf("A->operand= %s\n",ps->work->A->operand->label);
        else
                printf("A->operand= * NULL *\n");

        if(ps->work->B->operand != NULL)
                printf("B->operand= %s\n",ps->work->B->operand->label);
        else
                printf("B->operand= * NULL *\n");
        }
}
```

12.3. Simulator and Tools Program Listings.

*** NOTE ***
The following pages are the program listings for the thesis project. Due to printing capabilities of the hardware, an underscore is an undefined character. The ← character is the representation for the underscore.

```
/*+
** NAME:
**              gps-sim
**
** SYNOPSIS:
**              int gps-sim(root)
**
** PURPOSE:
**              Main routine for simulation phase of the compiler.  Pseudo
**              code has been generated, and we have to use that to create
**              the future and current events chain in the simulation.
**
** RETURNS:
**              Nothing
**
** MAINTENANCE:
**              29-Mar-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       CODE    *ps-cur;
globalref       CODE    *ps-root;
globalref       int     block-number;
globalref       int     strt-ptr;
globalref       INFO-STRT *strt[];

globaldef       BLOCK   *blk-stats= 0;          /* pointer to block statistics pointer */
globaldef       STOR    *stor-stats= 0;         /* pointer to storage statistics */
globaldef       QUE-STATS  *ptr-que= 0;         /* pointer to queue statistics */
globaldef       FAC-STATS  *fac-stats= 0;       /* pointer to facility statistics */
globaldef       FUT-EVENTS *ptr-fut= 0;         /* pointer to future events chain */
globaldef       CUR-EVENTS *ptr-cur= 0;         /* pointer to beginning of current events chain */
globaldef       CUR-EVENTS *end-cur= 0;         /* pointer to end of current events chain */
globaldef       int     abs-clock= 0;           /* absolute clock time */
globaldef       int     snap-interval= 0;       /* snap interval print (yes or no) */
globaldef       int     snap-time= 0;           /* snap interval print out time */

int gps-sim(root)
    SYM-TABLE   *root;
{
    CODE        *ps-work;                       /* working pointer to pseudo code */
    CUR-EVENTS  *wrk-cur;                       /* working pointer to current events chain */
    int         i, j;                           /* working variables */
    CUR-EVENTS  *simadv();
    CUR-EVENTS  *simbne();
    CUR-EVENTS  *simter();
    CUR-EVENTS  *siment();
    CUR-EVENTS  *simsez();

    crstor(root);                                                       /* create storage stati
**stics */
    for(i= 0 ; i < strt-ptr ; i++)                                      /* outer loop equal to
**number of start statements */
        {
        ptr-fut= ptr-cur= end-cur= NULL;                                /* init chain pointers
***/
        blk-stats= malloc(block-number * sizeof(BLOCK));                /* get room for statist
**ics */
        for(j= 0 ; j < block-number ; j++)                              /* loop equal to number
** of blocks */
                (blk-stats + j)->cur-contents= (blk-stats + j)->total= 0;

        for( ps-work= ps-root ; ps-work < ps-cur ; ps-work= ps-work->code-link )        /* loop equal t
**o number of pseudo code statements */
                {
                if( ps-work->operator == GENERATE )                     /* initial generate of
**transactions */
                        simgen(root,ps-work,abs-clock);
                }
        if(strt[i]->interval-print != 0)                                /* check for snap inter
**val printout */
                {
                snap-interval= YES;
```

```
                snap←time= strt[i]->interval←print;
                }
        while((strt[i]->sim←time != 0) && (wrk←cur != ERROR))                    /* loop equal to START
**A←operand */
                {
                trns←fut←cur(root);                                              /* put future onto curr
**ent events chain */
                wrk←cur= ptr←cur ;
                while((wrk←cur != NULL) && (strt[i]->sim←time != 0) && (wrk←cur != ERROR))/* circulate
**through current events chain */
                        {
                        if((snap←interval == YES) && (snap←time == 0))           /* should we print out
**?? */
                                {
                                report();                                        /* call report for prin
**tout */
                                snap←time= strt[i]->interval←print;               /* reinit snap time */
                                }
                        switch(wrk←cur->cur←block->code←link->operator)          /* check pseudo code op
**erator */
                                {
                                case(ADVANCE):
                                        wrk←cur= simadv(root,wrk←cur);
                                        break;

                                case(ASSIGN):
                                        simass(root,wrk←cur);
                                        break;

                                case(BR):
                                        simbr(root,wrk←cur);
                                        break;

                                case(BNE):
                                        wrk←cur= simbne(root,wrk←cur);
                                        break;

                                case(DEPART):
                                        simdep(root,wrk←cur);
                                        break;

                                case(ENTER):
                                        wrk←cur= siment(root,wrk←cur);
                                        break;

                                case(EQ):
                                case(LT):
                                case(GT):
                                        sim←compare(root,wrk←cur);
                                        break;

                                case(U):
                                case(NU):
                                        sim←gate←fac(root,wrk←cur);
                                        break;

                                case(SF):
                                case(SE):
                                case(SNF):
                                case(SNE):
                                        sim←gate←stor(root,wrk←cur);
                                        break;

                                case(LEAVE):
                                        simlev(root,wrk←cur);
                                        break;

                                case(NEG):
                                        simneg(wrk←cur);
                                        break;

                                case(RELEASE):
                                        simrel(wrk←cur);
                                        break;

                                case(QUEUE):
```

```
                                        simque(root,wrk←cur);
                                        break;

                                case(SEIZE):
                                        wrk←cur= simsez(root,wrk←cur);
                                        break;

                                case(TERMINATE):
                                        wrk←cur= simter(wrk←cur,i);
                                        break;
                                }
                        }
                }
        if(strt[i]->print == PRT)                                       /* check for printing o
**f report at end of run */
                report();
        }

}
```

```
/*+
** NAME:
**              simadv
**
** SYNOPSIS:
**              CUR←EVENTS *simadv(root,wrk←cur)
**                      SYM←TABLE       *root
**                      CUR←EVENTS      *wrk←cur
**
** PURPOSE:
**              Processes the ADVANCE statement.  Puts the advance on the
**              future events chain in a sorted order.  Calculates the block
**              departure time (BDT).  Takes the transaction off the current
**              events chain.
**
** RETURNS:
**              'New' pointer to current events chain.
**
** MAINTENANCE:
**              7-Apr-84        jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       BLOCK   *blk←stats;
globalref       int     abs←clock;

CUR←EVENTS *simadv(root,wrk←cur)
    SYM←TABLE   *root;
    CUR←EVENTS  *wrk←cur;
{
    CUR←EVENTS  *hold←ptr;              /* working pointer to the current events chain */
    FUT←EVENTS  *wrk←fut;               /* working pointer to the future events chain */
    float       simrnd();
    int         evaopr();

    (blk←stats + wrk←cur->blk←from)->cur←contents--;                                    /* let you into ADVANCE
** block */
    wrk←cur->cur←block= wrk←cur->cur←block->code←link;                                  /* get current get poin
**ters into ADVANCE */

    wrk←fut= wrk←cur;                                                                   /* transfer current eve
**nts pointer to future */
    wrk←fut->cur←block= wrk←cur->cur←block;
    hold←ptr= wrk←cur->bck←link;                                                        /* hold back link point
**er */
    if(wrk←cur->cur←block->A←operand->attr←pnt->lab←attribute == SNA←PARAM)     /* calculate BDT arriva
**l back on chain */
        wrk←fut->BDT= abs←clock + wrk←fut->param[evaopr(wrk←cur->cur←block->A←operand,root) - 1];
    else
        wrk←fut->BDT= abs←clock + evaopr(wrk←cur->cur←block->A←operand,root);
    wrk←fut->blk←from= wrk←fut->cur←block->block;                                       /* store block from whe
**nce you came */
    (blk←stats + wrk←fut->cur←block->block)->cur←contents++;                            /* get current statisti
**cs up-to-date */
    (blk←stats + wrk←fut->cur←block->block)->total++;

    remcur(wrk←cur);                                                                    /* remove transactio fr
**om the current events chain */
    srtfut(wrk←fut);                                                                    /* put transaction on t
**he sorted future events chain */

    return(hold←ptr);
}
```

```
/*+
** NAME:
**              simass
**
** SYNOPSIS:
**              int simass(root,wrk←cur)
**                      SYM←TABLE       *root
**                      CUR←EVENTS      *wrk←cur
**
** PURPOSE:
**              Processes the ASSIGN statement.  Puts the value of the assign
**              into the proper parameter.
**
** RETURNS:
**              Nothing
**
** MAINTENANCE:
**      10-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       BLOCK   *blk←stats;

int simass(root,wrk←cur)
    SYM←TABLE   *root;
    CUR←EVENTS  *wrk←cur;
{
    int         evaopr();

    (blk←stats + wrk←cur->blk←from)->cur←contents--;                        /* let you into ASSIGN
**block */
    wrk←cur->cur←block= wrk←cur->cur←block->code←link;                      /* get current get poin
**ters into ASSIGN */

    wrk←cur->param[(evaopr(wrk←cur->cur←block->A←operand,root) - 1)] + -     /* assign proper value
**to associated parameter */
                *(wrk←cur->cur←block->B←operand->attr←pnt->pnt←info);
    wrk←cur->blk←from= wrk←cur->cur←block->block;                           /* store block from whe
**nce you came */

    (blk←stats + wrk←cur->cur←block->block)->cur←contents++;                /* get block statistics
** up-to-date */
    (blk←stats + wrk←cur->cur←block->block)->total++;
}
```

```
/*+
** NAME:
**              simbne
**
** SYNOPSIS:
**              CUR-EVENTS *simbne(root,wrk-cur)
**                      SYM-TABLE       *root
**                      CUR-EVENTS      *wrk-cur;
**
** PURPOSE:
**              Processes the BNE statement.  The BNE is used in both the TEST
**              and the GATE statement.  This function processes the truth-bit
**              and will not allow you into the block if no label.
**
** RETURNS:
**              NULL            - if at end of current events chain
**              CUR-EVENTS *    - next working event on the events chain
**
** MAINTENANCE:
**              16-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       BLOCK   *blk-stats;
globalref       CODE    *ps-root;

CUR-EVENTS *simbne(root,wrk-cur)
    SYM-TABLE   *root;
    CUR-EVENTS  *wrk-cur;
{
    SYM-TABLE   *ret-status;            /* pointer to symbol table SEARCH */
    int         i;                      /* working variable */
    int         blk-from;               /* holder from block from whence you came */
    CODE        *wrk-pseudo;            /* working pseudo code pointer */
    CODE        *hld-pseudo;            /* holder for pointer to pseudo code */

    wrk-cur->cur-block= wrk-cur->cur-block->code-link;                          /* get pointer to BNE p
**seudo code */
    blk-from= wrk-cur->cur-block->block;                                        /* storre block form wh
**ence you came for later use */

    if(wrk-cur->truth-bit == YES)                                               /* if truth bit says ye
**s statement before true */
        {
        (blk-stats + wrk-cur->blk-from)->cur-contents--;                        /* let you into next bl
**ock */
        (blk-stats + wrk-cur->cur-block->block)->cur-contents++;                /* add the trans into b
**lock statistics */
        (blk-stats + wrk-cur->cur-block->block)->total++;                       /* both current and tot
**al numbers */
        wrk-cur->blk-from= blk-from;                                            /* get current block fr
**om whence you came */
        return(wrk-cur);
        }
    else
        {
        ret-status= strlab(root,wrk-cur->cur-block->A-operand->label,SEARCH);  /* get label, SEARCH sy
**mbol table */
        hld-pseudo= ret-status->attr-pnt->pnt-info;                             /* hold address of pseu
**do code for later */
        if(wrk-cur->cur-block->block != hld-pseudo->block)
                {
                (blk-stats + wrk-cur->blk-from)->cur-contents--;                /* let you into 'COMPAR
**E' block */
                (blk-stats + wrk-cur->cur-block->block)->cur-contents++;        /* add the trans into b
**lock statistics */
                (blk-stats + wrk-cur->cur-block->block)->total++;               /* both current and tot
**al numbers */
                }
        wrk-cur->blk-from= wrk-cur->cur-block->block;

        for(i= 0, wrk-pseudo= ps-root ;
                wrk-pseudo->block != (hld-pseudo->block - 1) ;
```

```
                wrk←pseudo= wrk←pseudo->code←link, i++)
                ;                                                       /* find statement befor
**e label */

        if(wrk←pseudo->block == wrk←pseudo->code←link->block)           /* make sure not a doub
**le block */
                wrk←cur->cur←block= wrk←pseudo->code←link;
        else
                wrk←cur->cur←block= wrk←pseudo;                         /* get current get poin
**ters into COMPARE */

        return(wrk←cur->bck←link);
        }

}
```

```
/*+
** NAME:
**              simbr
**
** SYNOPSIS:
**              int simbr(root,wrk←cur)
**                      SYM←TABLE       *root
**                      CUR←EVENTS      *wrk←cur;
**
** PURPOSE:
**              Processes the BR statement.  The BR statement is equivalent
**              to an unconditional TRANSFER for version 1.
**
** RETURNS:
**              Nothing
**
** MAINTENANCE:
**              7-Apr-84        jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       BLOCK   *blk←stats;
globalref       CODE    *ps←root;

int simbr(root,wrk←cur)
    SYM←TABLE   *root;
    CUR←EVENTS  *wrk←cur;
{
    SYM←TABLE   *ret←status;            /* return value from symbol table SEARCH */
    int         i;                      /* working value */
    int         blk←from;               /* holder from block from whence we came */
    CODE        *wrk←pseudo;            /* working pseudo code pointer */
    CODE        *hld←pseudo;            /* holder for pseudo code pointer */

    (blk←stats + wrk←cur->blk←from)->cur←contents--;                                /* let you into BRANCH
**block */
    wrk←cur->cur←block= wrk←cur->cur←block->code←link;                              /* get pointer into BRA
**NCH pseudo code */

    blk←from= wrk←cur->cur←block->block;                                            /* store the block from
** whence you came */
    (blk←stats + wrk←cur->cur←block->block)->cur←contents++;                        /* add the trans into b
**lock statistics */
    (blk←stats + wrk←cur->cur←block->block)->total++;                               /* both current and tot
**al numbers */

    ret←status= strlab(root,wrk←cur->cur←block->B←operand->label,SEARCH);           /* look for branching l
**abel */
    hld←pseudo= ret←status->attr←pnt->pnt←info;                                     /* retain this symbol t
**able value */

    for(i= 0, wrk←pseudo= ps←root ; wrk←pseudo->block != (hld←pseudo->block - 1) ; wrk←pseudo= wrk←pseu
**do->code←link, i++)
        ;                                                                           /* search for block bef
**ore the label to be branched to */
    if(wrk←pseudo->block == wrk←pseudo->code←link->block)                           /* make sure it is not
**a double block */
        wrk←cur->cur←block= wrk←pseudo->cur←block->code←link;
    else
        wrk←cur->cur←block= wrk←pseudo;                                             /* get current pointers
** to just before label to branch to */
    wrk←cur->blk←from= blk←from;                                                    /* put into block from
**whence you came */

}
```

```
/*+
** NAME:
**              simcmp
**
** SYNOPSIS:
**              int sim-compare(root,wrk-cur)
**                      SYM-TABLE       *root
**                      CUR-EVENTS      *wrk-cur
**
** PURPOSE:
**              Processes the EQ, LT, and GT statement.  This process tests for
**              the truth of that statement and if it is true sets the
**              truth-bit
**
** RETURNS:
**              Nothing.
**
** MAINTENANCE:
**      13-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       BLOCK   *blk-stats;

int sim-compare(root,wrk-cur)
    SYM-TABLE   *root;
    CUR-EVENTS  *wrk-cur;
{
    int         A-operator;             /* working A-operand evaluation */
    int         B-operator;             /* working B-operand evaluation */
    int         evaopr();

    if(wrk-cur->blk-from != wrk-cur->cur-block->block)
        (blk-stats + wrk-cur->blk-from)->cur-contents--;                    /* let you into 'COMPAR
**E' block */
    wrk-cur->cur-block= wrk-cur->cur-block->code-link;                      /* get current get poin
**ters into COMPARE */

    A-operator= evaopr(wrk-cur->cur-block->A-operand,root);                 /* evaluate A-operand *
**/
    if(wrk-cur->cur-block->A-operand->attr-pnt->lab-attribute == SNA-PARAM)
        A-operator= wrk-cur->param[A-operator - 1];

    B-operator= evaopr(wrk-cur->cur-block->B-operand,root);                 /* evaluate B-operand *
**/
    if(wrk-cur->cur-block->B-operand->attr-pnt->lab-attribute == SNA-PARAM)
        B-operator= wrk-cur->param[B-operator   1];

    if(wrk-cur->cur-block->operator == EQ)                                  /* check for equality o
**f A & B */
        {
        if(A-operator == B-operator)
                wrk-cur->truth-bit= YES;                                    /* equal YES */
        else
                wrk-cur->truth-bit= NO;                                     /* NOT equal */
        }
    else if(wrk-cur->cur-block->operator == LT)                             /* check for A less the
**n B */
        {
        if(A-operator < B-operator)
                wrk-cur->truth-bit= YES;                                    /* less than YES */
        else
                wrk-cur->truth-bit= NO;                                     /* NOT less than */
        }
    else if(wrk-cur->cur-block->operator == GT)                             /* check for A greater
**than B */
        {
        if(A-operator > B-operator)                                         /* greater than YES */
                wrk-cur->truth-bit= YES;
        else
                wrk-cur->truth-bit= NO;
        }
```

```
/*+
** NAME:
**              simdep
**
** SYNOPSIS:
**              int simdep(root,wrk←cur)
**                      SYM←TABLE       *root
**                      CUR←EVENTS      *wrk←cur
**
** PURPOSE:
**              Processes the DEPART statement.  The transaction leaves a queue
**              which it had entered in a previous QUEUE block
**
** RETURNS:
**              Nothing
**
** MAINTENANCE:
**      14-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       BLOCK   *blk←stats;
globalref       int     abs←clock;

int simdep(root,wrk←cur)
    SYM←TABLE   *root;
    CUR←EVENTS  *wrk←cur;
{
    int         hold;                   /* working variable */
    int         evaopr();

    (blk←stats + wrk←cur->blk←from)->cur←contents--;                            /* let you into DEPART
**block */
    wrk←cur->cur←block= wrk←cur->cur←block->code←link;                          /* get current get poin
**ters into DEPART */

    if(wrk←cur->time←que == abs←clock)                                          /* check for zero entry
** from queue */
        wrk←cur->ptr←que->zero←ent++;
    else
        wrk←cur->ptr←que->cum←time + (abs←clock - wrk←cur->time←que);

    if(wrk←cur->cur←block->B←operand == NULL)                                   /* depart default of 1
***/
        hold= 1;
    else if(wrk←cur->cur←block->B←operand->attr←pnt->lab←attribute == SNA←PARAM)
        hold= wrk←cur->param[(evaopr(wrk←cur->cur←block->B←operand,root) - 1)]; /* evaluate B←operand *
**/
    else
        hold= evaopr(wrk←cur->cur←block->B←operand,root);

    wrk←cur->ptr←que->cur←contents - = hold;                                    /* decrement queues cur
**rent contents by B←operand */

    wrk←cur->blk←from= wrk←cur->cur←block->block;                               /* store block from whe
'**nce you came */

    (blk←stats + wrk←cur->cur←block->block)->cur←contents++;                    /* get block stats up-t
**o-date */
    (blk←stats + wrk←cur->cur←block->block)->total++;
}
```

```
/*+
** NAME:
**              siment
**
** SYNOPSIS:
**              CUR←EVENTS *siment(root,wrk←cur)
**                      SYM←TABLE       *root
**                      CUR←EVENTS      *wrk←cur
**
** PURPOSE:
**              Processes the ENTER statement.  Lets a transaction take some
**              storage from the storage unit specified.
**
** RETURNS:
**              'NEW' current events chain pointer.
**
** MAINTENANCE:
**      13-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       BLOCK   *blk←stats;
globalref       int     abs←clock;

CUR←EVENTS *siment(root,wrk←cur)
    SYM←TABLE   *root;
    CUR←EVENTS  *wrk←cur;
{
    SYM←TABLE   *ret←status;            /* pointer to symbol table entry SEARCH */
    int         hold;                   /* working variable */
    int         no←room= NO;            /* block transaction from entering if no storage */
    int         evaopr();
    char        *stor←label;            /* storage label's name */

    if(wrk←cur->cur←block->code←link->A←operand->attr←pnt->lab←attribute == LAB←STOR)/* check for stora
**ge label */
        ret←status= wrk←cur->cur←block->code←link->A←operand;                   /* if so no need to do
**search */
    else
        {
        stor←label= evasto(root,wrk←cur->cur←block->code←link->A←operand,wrk←cur->cur←block->code←link)
**;
        if((ret←status= strlab(root,stor←label,SEARCH)) == ERROR)               /* get symbol table loc
**ation for storage */
                {
                printf("\nBlock %d\tAbsolute Clock %d\n",wrk←cur->cur←block->block,abs←clock);
                return(error(STR←NOT←FND));                                     /* storage undefined in
** program */
                }
        }

    if(wrk←cur->cur←block->code←link->B←operand == NULL)                        /* if B NULL default is
** 1 */
        {
        if(ret←status->attr←pnt->func←spec >= 1)                                /* check for room */
                (ret←status->attr←pnt->func←spec)--;                            /* if room decrement co
**unt */
        else
                no←room= YES;
        }
    else
        {
        hold= evaopr(wrk←cur->cur←block->code←link->B←operand,root);            /* evaluate B←operand *
**/
        if(wrk←cur->cur←block->code←link->B←operand->attr←pnt->lab←attribute == SNA←PARAM)
                {
                if(ret←status->attr←pnt->func←spec >= wrk←cur->param[hold])     /* is there room ? */
                        ret←status->attr←pnt->func←spec   - wrk←cur->param[hold];       /* yes room dec
**rement */
                else
                        no←room= YES;
                }
```

```
        else
                {
                if(ret->status->attr->pnt->func->spec >= hold)                  /* if room decrement to
**tal storage count */
                        ret->status->attr->pnt->func->spec - - hold;
                else
                        no->room= YES;
                }
        }

    if(no->room == NO)                                                          /* if there is room */
        {
        (blk->stats + wrk->cur->blk->from)->cur->contents--;                    /* let you into ENTER b
**lock */
        wrk->cur->cur->block= wrk->cur->cur->block->code->link;                 /* get current get poin
**ters into ENTER */

        wrk->cur->ptr->stor= fndstr(ret->status->label);                        /* find storage statist
**ics area */
        wrk->cur->time->stor= abs->clock;                                       /* fill in information
***/
        if(ret->status->attr->pnt->lab->attribute == SNA->PARAM)
                {
                wrk->cur->ptr->stor->cum->contents + - wrk->cur->param[hold];   /* cumulative contents
***/
                wrk->cur->ptr->stor->cur->cont + - wrk->cur->param[hold];       /* current contents */
                }
        else
                {
                wrk->cur->ptr->stor->cum->contents + - hold;
                wrk->cur->ptr->stor->cur->cont + - hold;
                }
        wrk->cur->ptr->stor->entries++;                                         /* total entries */
        if(wrk->cur->ptr->stor->cur->cont > wrk->cur->ptr->stor->max->cont)      /* check for maximum co
**ntents */
                wrk->cur->ptr->stor->max->cont= wrk->cur->ptr->stor->cur->cont;
        wrk->cur->blk->from= wrk->cur->cur->block->block;                       /* store block from whe
**nce you came */

        (blk->stats + wrk->cur->cur->block->block)->cur->contents++;            /* get block statistics
** up-to-date */
        (blk->stats + wrk->cur->cur->block->block)->total++;
        return(wrk->cur);
        }
    else
        return(wrk->cur->bck->link);
}
```

```
/*+
** NAME:
**              simfac
**
** SYNOPSIS:
**              int sim-gate-fac(root,wrk-cur)
**                      SYM-TABLE       *root
**                      CUR-EVENTS      *wrk-cur
**
** PURPOSE:
**              Processes the U, and NU statement.  This process tests for
**              the truth of that statement and if it is true sets the
**              truth-bit
**
** RETURNS:
**              Nothing.
**
** MAINTENANCE:
**      15-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

int sim-gate-fac(root,wrk-cur)
    SYM-TABLE   *root;
    CUR-EVENTS  *wrk-cur;
{
    int         A-operator;             /* working variable for A-operand evaluation */
    int         B-operator;             /* working variable for B-operand evaluation */
    FAC-STATS   *ptr-fac;               /* pointer to facility stats */
    int         evaopr();

    wrk-cur->cur-block= wrk-cur->cur-block->code-link;                          /* get current get poin
**ters into U or NU block */

    A-operator= evaopr(wrk-cur->cur-block->A-operand,root);                    /* evaluate A-operand *
**/
    if(wrk-cur->cur-block->A-operand->attr-pnt->lab-attribute == SNA-PARAM)
        A-operator= wrk-cur->param[A-operator   1];

    ptr-fac= fndfac(A-operator);                                               /* find associated faci
**lity */
    if(wrk-cur->cur-block->operator == U)                                      /* is facilty SEIZED ?
***/
        {
        if(ptr-fac->seized == YES)                                             /* facility is seized *
**/
                wrk-cur->truth-bit= YES;
        else                                                                   /* facility not seized
***/
                wrk-cur->truth-bit= NO;
        }
    else if(wrk-cur->cur-block->operator == NU)                                /* is facility not SEIZ
**ED */
        {
        if(ptr-fac->seized == NO)                                              /* facility is not seiz
.**ed */
                wrk-cur->truth-bit= YES;
        else                                                                   /* facility is seized *
**/
                wrk-cur->truth-bit= NO;
        }

}
```

```
/*+
** NAME:
**              sim-gate-stor
**
** SYNOPSIS:
**              int sim-gate-stor(root,wrk-cur)
**                      SYM-TABLE       *root
**                      CUR-EVENTS      *wrk-cur
**
** PURPOSE:
**              Processes the SE, SF, SNE, and SNF statement.  This process
**              tests for the truth of that statement and if it is true sets
**              the truth-bit
**
** RETURNS:
**              Nothing.
**
** MAINTENANCE:
**      16-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

int sim-gate-stor(root,wrk-cur)
    SYM-TABLE   *root;
    CUR-EVENTS  *wrk-cur;
{
    SYM-TABLE   *ret-status;            /* pointer to storage symbol table entry */
    STOR        *ptr-stor;              /* pointer to storage stats */
    char        *stor-label;            /* pointer to ascii label for storage */

    wrk-cur->cur-block= wrk-cur->cur-block->code-link;                          /* get current get poin
**ters into SE, SF, SNE, or SNF */

    if(wrk-cur->cur-block->A-operand->attr-pnt->lab-attribute == LAB-STOR)      /* if storage already n
**o need to find */
        {
        ret-status= wrk-cur->cur-block->A-operand;
        stor-label= ret-status->label;
        }
    else
        stor-label= evasto(root,wrk-cur->cur-block->A-operand,wrk-cur->cur-block);      /* get storage
**from symbol table */

    if((ret-status= strlab(root,stor-label,SEARCH)) == ERROR)
        return(error(STR-NOT-FND));                                             /* storage undefined */
**
    ptr-stor= fndstr(ret-status->label);                                        /* find storage area as
**sociated to trans */

    if(wrk-cur->cur-block->operator == SF)                                      /* storage full?? */
        {
        if(ptr-stor->cur-cont == ptr-stor->capacity)
                wrk-cur->truth-bit= YES;
        else
                wrk-cur->truth-bit= NO;
        }
    else if(wrk-cur->cur-block->operator == SE)                                 /* storage empty ?? */
        {
        if(ptr-stor->cur-cont == 0)
                wrk-cur->truth-bit= YES;
        else
                wrk-cur->truth-bit= NO;
        }
    else if(wrk-cur->cur-block->operator == SNF)                                /* storage not full ??
***/
        {
        if(ptr-stor->cur-cont < ptr-stor->capacity)
                wrk-cur->truth-bit= YES;
        else
                wrk-cur->truth-bit= NO;
        }
    else if(wrk-cur->cur-block->operator == SNE)                                /* storage not empty ??
```

```
** */
        {
        if(ptr->stor->cur->cont > 0)
                wrk->cur->truth->bit= YES;
        else
                wrk->cur->truth->bit= NO;
        }

}
```

```
/*+
** NAME:
**              simgen
**
** SYNOPSIS:
**              int simgen(root,ps←work,clock)
**                      SYM←TABLE       *root
**                      CODE            *ps←work
**                      int             clock
**
** PURPOSE:
**              Processes the GENERATE statement.  Puts the generate on the
**              future events chain in a sorted order.  Calculates the block
**              departure time (BDT).
**
** RETURNS:
**              Nothing.
**
** MAINTENANCE:
**              29-Mar-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

int simgen(root,ps←work,clock)
    SYM←TABLE   *root;
    CODE        *ps←work;
    int         clock;
{
    FUT←EVENTS  *wrk←fut;               /* working pointer to future events chain */
    int         i;                      /* working variable */
    int         A←value;                /* holding value for A←operand */
    int         B←value;                /* holding value for B←operand */
    float       temp←flt;               /* holder for floating calculation */
    float       simrnd();
    int         evaopr();

    wrk←fut= malloc(sizeof(FUT←EVENTS));                                    /* allocate space for f
**uture chain entry */
    wrk←fut->cur←block= ps←work;                                            /* point to pseudo code
** */
    for(i= 0 ; i < 10 ; i++)                                                /* init parameters */
        wrk←fut->param[i]= 0;

    if(ps←work->B←operand == NULL)                                          /* calculate block depa
**rture time */
        wrk←fut->BDT= clock + evaopr(ps←work->A←operand,root);
    else
        {
        A←value= evaopr(ps←work->A←operand,root);                           /* evaluate A←operand f
**ield */
        B←value= evaopr(ps←work->B←operand,root);                           /* evaluate B←operand f
**ield */
        if(ps←work->B←operand->attr←pnt->lab←attribute == CONSTANT)          /* round of kludge, to
**get even random distribution */
                {
                temp←flt= (B←value * simrnd()) + .5;
                if(simrnd() > .5)
                        temp←flt * - -1.;
                wrk←fut->BDT= clock + A←value + (int)temp←flt;
                }
        else
                wrk←fut->BDT= clock + (A←value * B←value);                  /* if B←operand non-CON
**STANT then it is multiplication time */
        }

    wrk←fut->blk←from= ps←work->block;                                      /* store block from whe
**nce you came */

    srtfut(wrk←fut);                                                        /* put new entry on to
**the sorted future events chain */

}
```

```
/*+
** NAME:
**              simlev
**
** SYNOPSIS:
**              int simlev(root,wrk←cur)
**                      SYM←TABLE       *root
**                      CUR←EVENTS      *wrk←cur
**
** PURPOSE:
**              Processes the LEAVE statement.  Lets a transaction give back
**              storage from the storage unit specified.
**
** RETURNS:
**              Nothing
**
** MAINTENANCE:
**      13-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       BLOCK   *blk←stats;
globalref       int     abs←clock;

int simlev(root,wrk←cur)
    SYM←TABLE   *root;
    CUR←EVENTS  *wrk←cur;
{
    SYM←TABLE   *ret←status;            /* storage symbol table pointer */
    int         hold;                   /* working variable */
    int         evaopr();
    char        *stor←label;            /* ascii storage label, name */

    (blk←stats + wrk←cur->blk←from)->cur←contents--;                            /* let you into LEAVE b
**lock */
    wrk←cur->cur←block= wrk←cur->cur←block->code←link;                          /* get current get poin
**ters into LEAVE */

    if(wrk←cur->cur←block->A←operand->attr←pnt->lab←attribute == LAB←STOR)      /* if already storage n
**o search necessary */
        ret←status= wrk←cur->cur←block->A←operand;
    else
        {
        stor←label= evasto(root,wrk←cur->cur←block->A←operand,wrk←cur->cur←block);
        if((ret←status= strlab(root,stor←label,SEARCH)) == ERROR)
                {
                printf("\nBlock %d\tAbsolute Clock %d\n",wrk←cur->cur←block->block,abs←clock);
                return(error(STR←NOT←FND));                                     /* error storage undefi
**ned */
                }
        }

    if(wrk←cur->cur←block->B←operand == NULL)                                   /* leave storage with d
**efault of 1 */
        (ret←status->attr←pnt->func←spec)++;
    else
        {
        hold= evaopr(wrk←cur->cur←block->B←operand,root);                       /* evaluate B←operand *
**/
        if(wrk←cur->cur←block->B←operand->attr←pnt->lab←attribute == SNA←PARAM) /* put statistical info
** into storage area */
                {
                ret←status->attr←pnt->func←spec + - wrk←cur->param[hold];
                wrk←cur->ptr←stor->cur←cont - - wrk←cur->param[hold];
                }
        else
                {
                ret←status->attr←pnt->func←spec + - hold;
                wrk←cur->ptr←stor->cur←cont - = hold;
                }
        }
```

```
    wrk←cur->blk←from= wrk←cur->cur←block->block;                    /* store block from whe
**nce you came */

    (blk←stats + wrk←cur->cur←block->block)->cur←contents++;         /* get block stats up-t
**o-date */
    (blk←stats + wrk←cur->cur←block->block)->total++;

    wrk←cur->ptr←stor->cum←tm + = (abs←clock - wrk←cur->time←stor);   /* reinit current event
**s pointer to storage */
    wrk←cur->time←stor= 0;
}
```

```
/*+
** NAME:
**              simneg
**
** SYNOPSIS:
**              int simneg(wrk←cur)
**                      CUR←EVENTS      *wrk←cur
**
** PURPOSE:
**              Processes the NEG statement.  Negates the value of an ASSIGN
**              and puts that value into a temporary location.
**
** RETURNS:
**              Nothing.
**
** MAINTENANCE:
**      10-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       8LOCK   *blk←stats;

int simneg(wrk←cur)
    CUR←EVENTS  *wrk←cur;
{
    FUT←EVENTS  *wrk←fut;               /* working pointer to future events chain */

    (blk←stats + wrk←cur->cur←block->block)->cur←contents--;                    /* let you into NEGATE
**block */
    wrk←cur->cur←block= wrk←cur->cur←block->code←link;                          /* get current get poin
**ters into NEGATE */

    wrk←cur->param[atoi(wrk←cur->cur←block->A←operand->attr←pnt->pnt←info)] +
                atoi(wrk←cur->cur←block->8←operand->attr←pnt->pnt←info);
    /* negate parameter */

    (blk←stats + wrk←fut->cur←block->block)->cur←contents++;                    /* add to block stats c
**urrent contents */
    (blk←stats + wrk←fut->cur←block->block)->total++;                           /* add to block stats t
**otal contents */
}
```

```
/*+
** NAME:
**              simque
**
** SYNOPSIS:
**              int simque(root,wrk←cur)
**                      SYM←TABLE       *root
**                      CUR←EVENTS      *wrk←cur
**
** PURPOSE:
**              Processes the QUEUE statement.  The transaction is accepted
**              from the previous block and entered in a queue
**
** RETURNS:
**              Nothing
**
** MAINTENANCE:
**      14-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       BLOCK   *blk←stats;
globalref       int     abs←clock;

int simque(root,wrk←cur)
    SYM←TABLE   *root;
    CUR←EVENTS  *wrk←cur;
{
    int         queue←name;             /* integer queue name */
    int         hold;                   /* working variable */
    int         evaopr();

    (blk←stats + wrk←cur->blk←from)->cur←contents--;                            /* let you into QUEUE b
**lock */
    wrk←cur->cur←block= wrk←cur->cur←block->code←link;                          /* get current get poin
**ters into QUEUE*/

    if(wrk←cur->cur←block->A←operand->attr←pnt->lab←attribute == SNA←PARAM)     /* evaluate queue name
***/
        queue←name= wrk←cur->param[evaopr(wrk←cur->cur←block->A←operand,root) - 1];
    else
        queue←name= evaopr(wrk←cur->cur←block->A←operand);
    wrk←cur->ptr←que= fndque(queue←name);                                       /* find queue statistic
**s block */
    wrk←cur->time←que= abs←clock;                                               /* store entry into que
**ue with current transaction */

    if(wrk←cur->cur←block->B←operand == NULL)                                   /* if B←operand NULL de
**fault is 1 */
        {
        wrk←cur->ptr←que->cur←contents++;                                       /* add to queues curren
**t contents */
        hold= 1;
        }
    else if(wrk←cur->cur←block->B←operand->attr←pnt->lab←attribute == SNA←PARAM)
        hold= wrk←cur->param[(evaopr(wrk←cur->cur←block->B←operand,root)   1)]; /* evaluate B←operand *
**/
    else
        hold= evaopr(wrk←cur->cur←block->B←operand,root);

    wrk←cur->ptr←que->cur←contents + = hold;                                    /* increase current con
**tents */
    wrk←cur->ptr←que->total←que + = hold;                                       /* increase cumulative
**contents */
    wrk←cur->ptr←que->total←ent++;                                              /* increase total entri
**es into queue */
    if(wrk←cur->ptr←que->cur←contents > wrk←cur->ptr←que->max←content)           /* check to see if max
**contents up to date */
        wrk←cur->ptr←que->max←content= wrk←cur->ptr←que->cur←contents;

    wrk←cur->blk←from= wrk←cur->cur←block->block;                               /* store block from whe
**nce you came */
```

```
    (blk->stats + wrk->cur->cur->block->block)->cur->contents++;                    /* get block statistics
** up-to-date */
    (blk->stats + wrk->cur->cur->block->block)->total++;
}
```

```
/*+
** NAME:
**              simrel
**
** SYNOPSIS:
**              int simrel(wrk-cur)
**                      CUR-EVENTS      *wrk-cur
**
** PURPOSE:
**              Processes the RELEASE statement.  Always lets a transaction
**              enter.
**
** RETURNS:
**              Nothing.
**
** MAINTENANCE:
**              15-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       BLOCK   *blk-stats;
globalref       int     abs-clock;

int simrel(wrk-cur)
    CUR-EVENTS  *wrk-cur;
{

    (blk-stats + wrk-cur->blk-from)->cur-contents--;                    /* let you into RELEASE
** block */
    wrk-cur->cur-block= wrk-cur->cur-block->code-link;                  /* get current get poin
**ters into RELEASE */

    wrk-cur->ptr-fac->seized= NO;                                       /* release facility */
    wrk-cur->ptr-fac->cum-time + = ( abs-clock - wrk-cur->time-sez );    /* add to the cumulativ
**e time */

    wrk-cur->blk-from= wrk-cur->cur-block->block;                       /* get block from whenc
**e you came */

    (blk-stats + wrk-cur->cur-block->block)->cur-contents++;            /* add current contents
** block statistics */
    (blk-stats + wrk-cur->cur-block->block)->total++;                   /* add to total content
**s block statistics */
}
```

```
/*+
** NAME:
**              simsez
**
** SYNOPSIS:
**              CUR←EVENTS *simsez(root,wrk←cur)
**                      SYM←TABLE       *root
**                      CUR←EVENTS      *wrk←cur
**
** PURPOSE:
**              Processes the SEIZE statement.  Lets a transaction enter a
**              facility if the facility is not being utilized.  Otherwise
**              it returns a NULL to put the transaction back on the CURRENT
**              EVENTS chain.
**
** RETURNS:
**              NULL            - if putting transaction back on the chain
**              CUR←EVENTS *    - the current events transaction just worked on
**
** MAINTENANCE:
**              15-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       BLOCK   *blk←stats;
globalref       int     abs←clock;

CUR←EVENTS *simsez(root,wrk←cur)
    SYM←TABLE   *root;
    CUR←EVENTS  *wrk←cur;
{
    FAC←STATS   *wrk←fac;               /* working pointer to facility statistics */
    int         hold;                   /* working variable */
    int         evaopr();

    hold= evaopr(wrk←cur->cur←block->code←link->A←operand,root);                /* evaluate A←operand *
**/
    if(wrk←cur->cur←block->code←link->A←operand->attr←pnt->lab←attribute == SNA←PARAM)
        hold= wrk←cur->param[ hold   1 ];
    wrk←fac= fndfac(hold);                                                      /* find facility statis
**tics associated with trans */

    if(wrk←fac->seized == NO)                                                   /* is facility seized ?
**? */
        {
        (blk←stats + wrk←cur->blk←from)->cur←contents--;                        /* if not enter SEIZE b
**lock */
        wrk←cur->cur←block= wrk←cur->cur←block->code←link;                      /* get current get poin
**ters into SEIZE */

        wrk←fac->seized= YES;                                                   /* set seize flag */
        wrk←cur->time←sez= abs←clock;                                           /* store entry time int
**o block */
        wrk←cur->ptr←fac= wrk←fac;                                              /* point to facility st
**ats within trans */
        wrk←cur->ptr←fac->total←ent++;                                          /* increment stats */
        wrk←cur->ptr←fac->seize←block= wrk←cur->cur←block->block;

        wrk←cur->blk←from= wrk←cur->cur←block->block;                           /* store block from whe
**nce you came */

        (blk←stats + wrk←cur->cur←block->block)->cur←contents++;                /* get block stats up-t
**o-date */
        (blk←stats + wrk←cur->cur←block->block)->total++;
        return(wrk←cur);
        }
    else
        return(wrk←cur->bck←link);
}
```

```
/*+
** NAME:
**              simter
**
** SYNOPSIS:
**              CUR-EVENTS *simter(wrk-cur,cur-run)
**                      CUR-EVENTS      *wrk-cur;
**                      int             cur-run;
**
** PURPOSE:
**              Processes the TERMINATE statement.  Eliminates the transaction
**              from the current events chain.  Decrements the transaction
**              termination count if the TERMINATE A-operand is not NULL.
**
** RETURNS:
**              Return next link n the CURRENT EVENTS chain.
**
** MAINTENANCE:
**              7-Apr-84        jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       BLOCK           *blk-stats;
globalref       INFO-STRT       *strt[];
globalref       int             snap-interval;
globalref       int             snap-time;

CUR-EVENTS *simter(wrk-cur,cur-run)
    CUR-EVENTS  *wrk-cur;
    int         cur-run;                /* current run of this program */
{
    CUR-EVENTS  *hold-ptr;

    (blk-stats + wrk-cur->blk-from)->cur-contents--;                        /* you are let into ter
**minate */
    wrk-cur->cur-block= wrk-cur->cur-block->code-link;                       /* get current block -
**TERMINATE */

    (blk-stats + wrk-cur->cur-block->block)->total++;                       /* both current and tot
**al numbers */

    if(wrk-cur->cur-block->A-operand != NULL)                               /* if isn't NULL termin
**ate the run count */
        {
        strt[cur-run]->sim-time - - *(wrk-cur->cur-block->A-operand->attr-pnt->pnt-info);
        if(snap-interval == YES)
                snap-time   = *(wrk-cur->cur-block->A-operand->attr-pnt->pnt-info);
        }

    hold-ptr= wrk-cur->bck-link;                                            /* go onto next current
** event */
    wrk-cur->blk-from= wrk-cur->cur-block->block;                            /* get block from whenc
**e you came */

    remcur(wrk-cur);
    free(wrk-cur);                                                          /* free the contents */
.**

    return(hold-ptr);
}
```

```
/*+
** NAME:
**              report
**
** SYNOPSIS:
**              int report()
**
** PURPOSE:
**              Standard output of the GPSS simulation.
**
** RETURNS:
**              Not applicable
**
** MAINTENANCE:
**              28-Mar-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       int     abs_clock;
globalref       int     block_number;
globalref       BLOCK   *blk_stats;
globalref       STOR    *stor_stats;
globalref       QUE_STATS *ptr_que;
globalref       FAC_STATS *fac_stats;

int report()
    {
    int         i, j, k;                /* working variables for FOR loops */
    float       flt_hold;               /* worker for floating point arithmetic */
    STOR        *wrk_stor;              /* working storage stats pointer */
    QUE_STATS   *wrk_que;               /* working queue stats pointer */
    FAC_STATS   *wrk_fac;               /* working facility stats pointer */

    k= 0;
    printf("\n\nAbsolute clock:\t%d\n",abs_clock);
    for(j= block_number ; j > 0 ; j= j-30)                              /* print blocok statist
**ics */
        {
        printf("\nBLOCK COUNTS\n");
        printf("Block\tCurrent\tTotal\tBlock\tCurrent\tTotal\tBlock\tCurrent\tTotal\t\n");
        printf("=====\t=======\t=====\t=====\t=======\t=====\t=====\t=======\t=====\t\n");
        for(i= 0 ; (i < 10) && (j > i) ; i++)
                {
                if((i + 20) < j)
                        printf("\n%5d\t%7d\t%5d\t%5d\t%7d\t%5d\t%5d\t%7d\t%5d",
                                (i + k + 1),(blk_stats + k+ i)->cur_contents,(blk_stats + k + i)->total
**,
                                (i + k + 11),(blk_stats + 10 + k + i)->cur_contents,(blk_stats + 10 + k
** + i)->total,
                                (i + k + 21),(blk_stats + 20 + k + i)->cur_contents,(blk_stats + 20 + k
** + i)->total);
                else if((i + 10) < j)
                        printf("\n%5d\t%7d\t%5d\t%5d\t%7d\t%5d",(i+ k + 1),
                                (blk_stats + k + i)->cur_contents,(blk_stats + k + i)->total,
                                (i + k + 11),(blk_stats + 10 + k + i)->cur_contents,
                                (blk_stats + 10 + k + i)->total);
                else
                        printf("\n%5d\t%7d\t%5d",(i + k + 1),(blk_stats + k + i)->cur_contents,
                                (blk_stats + k + i)->total);
                }
        k + = 30;
        }

    if(fac_stats != NULL)                                               /* print facility stati
**stics if pointer not NULL */
        {
        printf("\n\nFacility\tAverage\t\tNumber\tAverage\t\tSeizing\n");
        printf("\t\tUtilization\tEntries\tTime/Tran\tTrans.No.\n");
        for(wrk_fac= fac_stats ; wrk_fac != NULL ; wrk_fac= wrk_fac->link)
                {
                flt_hold= ((float)wrk_fac->cum_time/(float)wrk_fac->total_e
                printf("%8d\t%11f    %7d    %9f\t%9d\n",wrk_fac->name,
```

```
                        (((float)wrk->fac->total->ent/(float)abs->clock) * flt->hold),
                        wrk->fac->total->ent,flt->hold,(wrk->fac->seize->block + 1));
                }
        }

    if(ptr->que != NULL)                                        /* print queue stats if
** pointer not NULL */
        {
        printf("\nQueue   Maximum   Average   Total   Zero    Average     Current\n");
        printf("\tContents  Contents  Entries Entries Time/Trans  Contents\n");
        for(wrk->que= ptr->que ; wrk->que != NULL ; wrk->que= wrk->que->link)
                {
                flt->hold= ((float)wrk->que->cum->time/(float)wrk->que->total->ent);
                printf("%5d  %8d   %8f  %5d %7d %10f%8d\n",wrk->que->name,wrk->que->max->content,
                        (((float)wrk->que->cum->contents/(float)abs->clock) * flt->hold),
                        wrk->que->total->que,wrk->que->zero->ent,flt->hold,
                        wrk->que->cur->contents);
                }
        }

    if((stor->stats != NULL) && (stor->stats->entries != 0))    /* print storage stats
**if there have been entries */
        {
        printf("\nStorage  Capacity   Average   Average    Entries  Average    Current   Maximum\n");
        printf("\t\t    Contents  Utilization \t  Time/Trans Contents  Contents\n");
        for(wrk->stor= stor->stats ; wrk->stor != NULL ; wrk->stor= wrk->stor->link)
                {
                flt->hold= (float)wrk->stor->cum->contents * ((float)wrk->stor->cum->tm/(float)wrk->stor->ent
**ries);
                flt->hold= flt->hold / (float)abs->clock;
                printf("%s\t%8d    %8f  %7f   %7d   %7f  %7d  %7d\n",wrk->stor->stor->nam,wrk->stor->capac
**ity,
                flt->hold,(flt->hold/(float)wrk->stor->capacity),wrk->stor->cum->contents,
                ((float)wrk->stor->cum->tm/(float)wrk->stor->entries),
                wrk->stor->cur->cont,wrk->stor->max->cont);
                }
        }

    }
```

```
/*+
** NAME:
**              addcur
**
** SYNOPSIS:
**              int addcur(wrk←cur)
**                      CUR←EVENTS      *wrk←cur;
**
** PURPOSE:
**              Add new transaction to the current events chain.
**
** RETURNS:
**              Nothing
**
** MAINTENANCE:
**              7-Apr-84        jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       CUR←EVENTS *ptr←cur;
globalref       CUR←EVENTS *end←cur;

int addcur(wrk←cur)
    CUR←EVENTS  *wrk←cur;
{

    if(ptr←cur == NULL)                                         /* add to an empty chai
**n */
        {
        end←cur= ptr←cur= wrk←cur;                              /* easy addition, chang
**e pointers */
        wrk←cur->for←link= wrk←cur->bck←link= NULL;              /* make links initially
** NULL */
        }
    else
        {
        wrk←cur->bck←link= end←cur->bck←link;                   /* correct the end of t
**he chain */
        end←cur->bck←link= wrk←cur;                             /* make old end point t
**o new */
        wrk←cur->for←link= end←cur;                             /* make new end point t
**o old */
        end←cur= wrk←cur;                                       /* change end pointer o
**f current events chain */
        }
}
```

```
/*+
** NAME:
**              crstor
**
** SYNOPSIS:
**              crstor(root)
**                      SYM-TABLE       *root;
**
** PURPOSE:
**              Create storage statistics.
**
** RETURNS:
**              Not applicable.
**
** MAINTENANCE:
**              14-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       STOR    *stor-stats;

int crstor(root)
    SYM-TABLE   *root;
    {
    STOR        *wrk-stor;

    if(root->label == NULL)
        return;

    if((root->attr-pnt->lab-attribute) == LA8-STOR)                         /* is label attribute a
** storage */
        {
        if(stor-stats == NULL)                                              /* if stor-stats is non
**existant */
                wrk-stor= stor-stats= malloc(sizeof(STOR));                 /* create areas */
        else
                {
                for(wrk-stor= stor-stats ; wrk-stor->link != NULL ; wrk-stor= wrk-stor->link)
                        ;
                wrk-stor->link= malloc(sizeof(STOR));                       /* link in storage area
**s stats */
                wrk-stor= wrk-stor->link;
                }
        wrk-stor->stor-nam= root->label;                                    /* put information into
** storage stats area */
        wrk-stor->capacity= root->attr-pnt->leng-info;
        }

    if(root->left != NULL)                                                  /* look for storage sta
**ts down left side */
        crstor(root->left);

    if(root->right != NULL)                                                 /* look for storage sta
**ts down right side */
        crstor(root->right);
'   }
```

```
/*+
** NAME:
**              evafun
**
** SYNOPSIS:
**              int evafun(operand)
**                      SYM-TABLE       *operand
**
** PURPOSE:
**              Evaluation of the FUNCTION.
**
** RETURNS:
**              Integer value of the FUNCTION.
**
** MAINTENANCE:
**              29-Mar-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

int evafun(operand)
    SYM-TABLE   *operand;
{
    int         i= 0;                   /* working variable */
    float       temp-flt;               /* floating holder */
    float       simrnd();

    temp-flt= simrnd();                                                 /* get random number */
**
    for( i= 0 ; operand->attr-pnt->leng-info > i ; i++)                 /* look through the fun
**ction data for correct range */
        {
        if( temp-flt < *(operand->attr-pnt->flo-num + i))
                return(*(operand->attr-pnt->pnt-info + i));
        }                                                               /* return correct integ
**er data */

return(*(operand->attr-pnt->pnt-info + (i-1)));
}
```

```
/*+
** NAME:
**              evaopr
**
** SYNOPSIS:
**              int evaopr(operand,root)
**                      SYM←TABLE       *operand
**                      SYM←TABLE       *root
**
** PURPOSE:
**              Evaluation of the operands for the simulator.
**
** RETURNS:
**              Integer value of the operand.
**
** MAINTENANCE:
**              29-Mar-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

int evaopr(operand,root)
    SYM←TABLE   *operand;
    SYM←TABLE   *root;
{

    if(operand->attr←pnt->lab←attribute == CONSTANT)                        /* if label attribute i
**s a CONSTANT */
        return(*(operand->attr←pnt->pnt←info));
    else if(operand->attr←pnt->lab←attribute == LAB←VAR)                    /* if label attribute i
**s a VARIABLE */
        return(evavar(operand,root,((int)operand->attr←pnt->pnt←info + (int)operand->attr←pnt->leng←inf
**o)));
    else if(operand->attr←pnt->lab←attribute == LAB←FUNC)                   /* if label attribute i
**s a FUNCTION */
        return(evafun(operand));
    else if(operand->attr←pnt->lab←attribute == SNA←PARAM)                  /* if label attribute i
**s a PARAMETER */
        return(*(operand->attr←pnt->pnt←info));
}
```

```
/*+
** NAME:
**              evasto
**
** SYNOPSIS:
**              int evasto(root,operand,wrk←cur)
**                      SYM←TABLE       *root
**                      SYM←TABLE       *operand
**                      CUR←EVENTS      *wrk←cur
**
** PURPOSE:
**              Evaluation of the storage A←operand.
**
** RETURNS:
**              Ascii representation of the storage label.
**
** MAINTENANCE:
**              13-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

char *evasto(root,operand,wrk←cur)
    SYM←TABLE   *root;
    SYM←TABLE   *operand;
    CUR←EVENTS  *wrk←cur;
{
    static      string[80];             /* area used for decimal to ascii conversion */
    int         hold;                   /* working hold variable */
    int         evaopr();

    if((operand->attr←pnt->lab←attribute == LAB←VAR) ||
       (operand->attr←pnt->lab←attribute == LAB←FUNC) ||
       (operand->attr←pnt->lab←attribute == CONSTANT))
       hold= evaopr(operand,root);                                          /* evaluate operand and
** store in hold */
    else if(operand->attr←pnt->lab←attribute == SNA←PARAM)
       hold= wrk←cur->param[evaopr(operand,root) - 1];                      /* put value of paramet
**er into hold area */

    sprintf(string,"%d",hold);                                              /* convert decimal to a
**scii string */
    if(hold < 10)                                                           /* NULL terminate */
       string[1]= NULL;
    else if(hold < 100)
       string[2]= NULL;
    else if(hold < 1000)
       string[3]= NULL;

    return(&string);
}
```

```
/*+
** NAME:
**              evavar
**
** SYNOPSIS:
**              int evavar(operand,root,leng-info)
**                      SYM-TABLE       *operand
**                      SYM-TABLE       *root
**                      char            *leng-info
**
** PURPOSE:
**              Evaluation of a VARIABLE operand.
**
** RETURNS:
**              Integer value of the VARIABLE.
**
** MAINTENANCE:
**              29-Mar-84       jvd     created
**
-*/
#include        stdio
#include        ctype
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

int evavar(operand,root,leng-info)
    SYM-TABLE   *operand;
    SYM-TABLE   *root;
    char        *leng-info;             /* information pertaining to length of VARIABLE string */
{
    char        *str-info;              /* working pointer to variable string */
    char        string[80];
    SYM-TABLE   *label;                 /* put symbol table pointer into pointer */
    int         A-operand;              /* holder for A-operand's results */
    int         B-operand;              /* holder for B-operand's results */
    char        *fndcol();              /* find colon of the variable string */

    str-info= operand->attr-pnt->pnt-info;                                      /* initialize pointer *
**/

    str-info= fndcol(string,str-info);                                          /* find first ':' */

    if(isdigit(string[0]))                                                      /* change ascii string
**into integer */
        A-operand= atoi(string);
    else
        {
        label= strlab(root,string,SEARCH);                                      /* find variable within
** variable */
        if(label->attr-pnt->lab-attribute == LAB-FUNC)                          /* evaluate A-operand *
**/
                A-operand= evafun(label);
        else if((label->attr-pnt->lab-attribute == TEMP-VAR) ||
                (label->attr-pnt->lab-attribute == LAB-VAR))
                A-operand= evavar(label,root,((int)label->attr-pnt->pnt-info + (int)label->attr-pnt->le
**ng-info));
        }

    if(strcmp(str-info,"~") == 0)                                               /* check for a negate *
**/
        {
        A-operand = - A-operand;                                                /* negate A-operand */
        str-info= fndcol(string,str-info);                                      /* find next colon */
        }

    while(str-info < leng-info)                                                 /* circulate through th
**e string */
        {
        str-info= fndcol(string,str-info);                                      /* find next colon */
        if(isdigit(string[0]))                                                  /* evaluate the ascii s
**tring */
                B-operand= atoi(string);
        else
                {
                label= strlab(root,string,SEARCH);                              /* find variable within
```

```
** variable */
                if(label->attr←pnt->lab←attribute == LAB←FUNC)                  /* evaluate B←operand *
**/
                        B←operand= evafun(label);
                else if((label->attr←pnt->lab←attribute == TEMP←VAR) ||
                        (label->attr←pnt->lab←attribute == LAB←VAR))
                        B←operand= evavar(label,root,((int)label->attr←pnt->pnt←info + (int)label->attr
**←pnt->leng←info));
                }

        if(strcmp(str←info,"~") == 0)                                           /* should we negate */
                {
                B←operand = - B←operand;                                        /* negate B←operand */
                str←info= fndcol(string,str←info);                              /* find next colon */
                }

        str←info= fndcol(string,str←info);                                      /* find next colon */

        if(string[0] == ADDITION)                                               /* is the operand addit
**ion ? */
                A←operand + = B←operand;
        else if( string[0] == SUBTRACTION)                                      /* is the operator subt
**raction ?? */
                A←operand - = B←operand;
        else if(string[0] == MULTIPLICATION)                                    /* is the operator mult
**iplication ??? */
                A←operand * = B←operand;
        else if(string[0] == DIVISION)                                          /* is the operator divi
**sion ???? */
                A←operand / = B←operand;
    }

return(A←operand);
}
```

```
/*+
** NAME:
**              fndcol
**
** SYNOPSIS:
**              int fndcol(operand,pnt←info)
**                      char    *operand;
**                      char    *pnt←info;
**
** PURPOSE:
**              Find colon when evaluating variable string, and copy the
**              string into the location specified by the input operand.
**
** RETURNS:
**              ponter to the VARIABLE string nformation
**
** MAINTENANCE:
**              29-Mar-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

int fndcol(operand,pnt←info)
    char        *operand;
    char        *pnt←info;
{
    int         ret←status;

    if((ret←status= strcspn(pnt←info,":")) == 0)                    /* use facility to find
** the colon */
        {
        printf("\nProgrammer goof up\n");
        return(ERROR);
        }

    strncpy(operand,pnt←info,ret←status);                           /* copy ascii string be
**tween colons */
    operand[ret←status]= NULL;                                      /* into space provided
**and NULL terminate */

    pnt←info = pnt←info + (ret←status + 1);                         /* increase string poin
**ter past ascii string */

return(pnt←info);
}
```

```
/*+
** NAME:
**              fndfac
**
** SYNOPSIS:
**              FAC←STATS *fndfac(fac←label)
**                      int     fac←label
**
** PURPOSE:
**              Find the facility associated with this transaction.
**
** RETURNS:
**              Pointer to facility statistics associated with this transaction.
**
** MAINTENANCE:
**              15-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       FAC←STATS       *fac←stats;

FAC←STATS *fndfac(fac←label)
    int fac←label;
    {
    FAC←STATS   *wrk←fac;               /* working facilty pointer */
    FAC←STATS   *new←fac;               /* new facility pointer */

    if(fac←stats != NULL)                                                   /* is there a facilitie
**s stats area */
        {
        for(wrk←fac= fac←stats ; wrk←fac != NULL ; wrk←fac= wrk←fac->link)
                {                                                           /* look through stats f
**or a match */
                if(fac←label == wrk←fac->name)
                        break;
                }

        if(wrk←fac == NULL)                                                 /* end of stats list *
**
                {
                new←fac= malloc(sizeof(FAC←STATS));                         /* create new facility
**stats */
                new←fac->link= NULL;                                        /* fill in information
***/
                new←fac->name= fac←label;
                for(wrk←fac= fac←stats ; wrk←fac->link != NULL ; wrk←fac= wrk←fac->link)
                        ;
                wrk←fac->link= new←fac;
                wrk←fac->seized= NO;
                wrk←fac= new←fac;
                }
    }
    else
        {
        fac←stats= malloc(sizeof(FAC←STATS));                               /* empty stats list '
        fac←stats->link= NULL;                                              /* create stats area
**
        fac←stats->name= fac←label;
        wrk←fac= fac←stats;
        }

    return(wrk←fac);
    }
```

```
/*+
** NAME:
**              fndque
**
** SYNOPSIS:
**              QUE-STATS *fndque(que-label)
**                      int     que-label
**
** PURPOSE:
**              Find the QUEUE associated with this transaction.  If not created
**              then create the queue statistic block.
**
** RETURNS:
**              Pointer to queue statistics associated with this transaction.
**
** MAINTENANCE:
**              14-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       QUE-STATS       *ptr-que;

QUE-STATS *fndque(que-label)
    int que-label;
    {
    QUE-STATS   *wrk-que;               /* working pointer to queue stats */
    QUE-STATS   *new-que;               /* pointer to new queue stats */

    if(ptr-que != NULL)                                                 /* find queue stats whi
**ch matches que-label */
        {
        for(wrk-que= ptr-que ; wrk-que != NULL ; wrk-que= wrk-que->link)
                {
                if(wrk-que->name == que-label)
                        break;
                }
        if(wrk-que == NULL)                                             /* if NULL not found */
**
                {
                new-que= malloc(sizeof(QUE-STATS));                     /* create new queue sta
**ts */
                new-que->link= NULL;                                    /* fill with informatio
**n */
                new-que->name= que-label;
                for(wrk-que= ptr-que ; wrk-que->link != NULL ; wrk-que= wrk-que->link)
                        ;
                wrk-que->link= new-que;
                wrk-que= new-que;
                }
        }
    else
        {                                                               /* no queue stats */
        ptr-que= malloc(sizeof(QUE-STATS));                             /* create initial entry
** */
        ptr-que->link= NULL;
        ptr-que->name= que-label;
        wrk-que= ptr-que;
        }

    return(wrk-que);
    }
```

```
/*+
** NAME:
**              fndstr
**
** SYNOPSIS:
**              STOR *fndstr(stor←label)
**                      char    *stor←label
**
** PURPOSE:
**              Find the storage associated with this transaction.
**
** RETURNS:
**              Pointer to storage statistics associated with this transaction.
**
** MAINTENANCE:
**              14-Apr-84       jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       STOR    *stor←stats;

STOR *fndstr(stor←label)
    char        *stor←label;
    {
    STOR        *wrk←stor;      /* working storage pointer */

    for(wrk←stor= stor←stats ; wrk←stor != NULL ; wrk←stor= wrk←stor->link)
        {
        if(strcmp(wrk←stor->stor←nam,stor←label) == 0)
                break;
        }                                                               /* find storage stats
**/

    if(wrk←stor == NULL)                                                /* if NULL no storage
**hus storage undefined */
        return(error(STR←NOT←FND));

    return(wrk←stor);
    }
```

```
/*+
** NAME:
**              remcur
**
** SYNOPSIS:
**              int remcur(wrk←cur)
**                      CUR←EVENTS      *wrk←cur;
**
** PURPOSE:
**              Removes a transaction from the current events chain
**
** RETURNS:
**              Nothing
**
** MAINTENANCE:
**              7-Apr-84        jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       CUR←EVENTS      *ptr←cur;
globalref       CUR←EVENTS      *end←cur;

int remcur(wrk←cur)
    CUR←EVENTS  *wrk←cur;
{
    if((wrk←cur == ptr←cur) && (ptr←cur == end←cur))                    /* only one entry so re
**move it */
        ptr←cur= end←cur= NULL;
    else if((wrk←cur == ptr←cur) && (ptr←cur != end←cur))                /* is it beginning of c
**urrent events chain */
        {
        ptr←cur->bck←link->for←link= ptr←cur->for←link;
        ptr←cur= ptr←cur->bck←link;
        }
    else                                                                /* remove entry from mi
**ddle */
        {
        wrk←cur->for←link->bck←link= wrk←cur->bck←link;                  /* unlink the transacti
**on from the chain */
        if(wrk←cur->bck←link != NULL)
                wrk←cur->bck←link->for←link= wrk←cur->for←link;          /* both forward and bac
**kward */
        else
                end←cur= wrk←cur->for←link;
        }
}
```

```
/*+
** NAME:
**              simrnd
**
** SYNOPSIS:
**              float simrnd()
**
** PURPOSE:
**              Create a random number in the range from 0.0 - 1.0.
**
** RETURNS:
**              Floating number in the range from 0.0 - 1.0.
**
** MAINTENANCE:
**              29-Mar-84       jvd     created
**
**
-*/

#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

float simrnd()
    {
    int random;                         /* integer random number from utility */
    float temp;                         /* holder for random in floating point */
    float flrand;                       /* the random number between 0 - 1 */
    int rand();

    random= rand();                     /* use VAX random number facility */
    temp= random;                       /* place integer into float for conversion */
    flrand= temp / 2147483647.;         /* convert to range 0.0 - 1.0 */

    return(flrand);
    }
```

```
/*+
** NAME:
**              srtfut
**
** SYNOPSIS:
**              int srtfut(wrk←fut)
**                      FUT←EVENTS      *wrk←fut
**
** PURPOSE:
**              Sort future events chain: add new BDT
**
** RETURNS:
**              Nothing
**
** MAINTENANCE:
**              2-Apr-84        jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       FUT←EVENTS *ptr←fut;

int srtfut(wrk←fut)
    FUT←EVENTS  *wrk←fut;
{
    FUT←EVENTS  *i;             /* working pointer to facility stats */

    if(ptr←fut == NULL)                                                 /* is there nothing on
**the future events chain */
        {
        wrk←fut->for←link= NULL;                                        /* easy link NULL termi
**nate */
        wrk←fut->bck←link= NULL;                                        /* both forward and bac
**kward links */
        ptr←fut= wrk←fut;                                               /* place addition to ch
**ain in FUT←EVENTS */
        }
    else
        {
        for(i= ptr←fut ; wrk←fut->BDT > i->BDT ; i= i->bck←link)        /* find area to enter *
**/
                {
                if(i->bck←link == NULL)
                        break;
                }
        if((wrk←fut->BDT > i->BDT) && (i->bck←link == NULL))            /* check to make sure e
**nd of chain to add to */
                {
                wrk←fut->bck←link= i->bck←link;                         /* change backlink to n
**ew arrival */
                i->bck←link= wrk←fut;                                   /* change previous back
** link */
                wrk←fut->for←link= i;                                   /* point new end to pre
**vious end link */
                }
        else if(i->for←link == NULL)                                    /* your at beginning of
,** chain */
                {
                wrk←fut->for←link= i->for←link;                         /* point to old forward
** link */
                wrk←fut->bck←link= i;                                   /* point to old beginni
**ng */
                i->for←link= wrk←fut;                                   /* correct backwards li
**nk for entry */
                ptr←fut= wrk←fut;                                       /* change global pointe
**r to new beginning */
                }
        else
                {
                wrk←fut->bck←link= i->for←link->bck←link;               /* put back link NULL i
**nto place */
                wrk←fut->for←link= i->for←link;                         /* point to old ending
**of chain */
```

```
                i->for_link->bck_link= wrk_fut;                    /* point old en
**d to new end of chain */
                i->for_link= wrk_fut;
                }
    }

}
```

```
/*+
** NAME:
**              trns-fut-cur
**
** SYNOPSIS:
**              int trns-fut-cur(root)
**                      SYM-TABLE       *root;
**
** PURPOSE:
**              Add events onto the current events chain from the future
**              events chain.
**
** RETURNS:
**              Nothing
**
** MAINTENANCE:
**              7-Apr-84        jvd     created
**
-*/
#include        stdio
#include        "[vandellon.thesis]gpss"
#include        "[vandellon.thesis]simulator"

globalref       FUT-EVENTS *ptr-fut;
globalref       BLOCK      *blk-stats;
globalref       int     abs-clock;

int trns-fut-cur(root)
    SYM-TABLE   *root;
{
    FUT-EVENTS  *temp-fut;              /* working future events pointer */
    CUR-EVENTS *wrk-cur;                /* working current events pointer */

    abs-clock= ptr-fut->BDT;                                                    /* update the clock to
**next BDT */

    while((ptr-fut != NULL) && (ptr-fut->BDT <= abs-clock))                     /* do you have chain an
**d is it less then abs-clock */
        {
        wrk-cur= ptr-fut;
        if(ptr-fut->cur-block->operator != ADVANCE)
                {
                (blk-stats + ptr-fut->cur-block->block)->total++;               /* ADVANCE came off fut
**ure events */
                (blk-stats + ptr-fut->cur-block->block)->cur-contents++;        /* so don't need to rea
**dd */
                }
        temp-fut= ptr-fut;

        if(ptr-fut->bck-link != NULL)
                ptr-fut->bck-link->for-link= ptr-fut->for-link;                 /* unlink future events
** chain */
        ptr-fut= ptr-fut->bck-link;

        if(temp-fut->cur-block->operator ==  GENERATE)
                simgen(root,temp-fut->cur-block,temp-fut->BDT);                 /* generate the next ev
**ent on the current events chain */

        addcur(wrk-cur);                                                        /* add to current event
**s chain
        }

}
```

12.4. # *include* Files and Link File.

*** NOTE ***
The following pages are the program listings for the thesis project. Due to printing capabilities of the hardware, an underscore is an undefined character. The ← character is the representation for the underscore.

```
/*+
** NAME:
**              gpss.h  - #include file containing constants for the GPSS
**                        compiler
**
** SYNOPSIS:
**              #include        gpss
**
** PURPOSE:
**              #include file containing constants for the GPSS compiler.
**
** RETURNS:
**              Not applicable.
**
** MAINTENANCE:
**              7-June-83       jvd     created
**
**
-*/

#define SUCCESS         0
#define ERROR           -1
#define ON              1
#define OFF             0
#define YES             1
#define NO              0
#define MAXLINE         132
#define NUMB←KEY←WORDS  19
#define EOL             -1
#define COMMA           -2

struct keyword
    {
    char        *keyword;
    }keyarray[NUMB←KEY←WORDS]=
        {
        "ADVANCE",
        "ASSIGN",
        "DEPART",
        "END",
        "ENTER",
        "FUNCTION",
        "GATE",
        "GENERATE",
        "LEAVE",
        "QUEUE",
        "RELEASE",
        "SEIZE",
        "SIMULATE",
        "START",
        "STORAGE",
        "TERMINATE",
        "TEST",
        "VARIABLE",
        "TRANSFER"
        };

/*
**      Numeric equivalents for GPSS commands
*/

#define SIMULATE        1
#define START           2
#define END             3
#define STORAGE         4
#define VARIABLE        5
#define FUNCTION        6
#define GENERATE        7
#define TERMINATE       8
#define ADVANCE         9
#define ASSIGN          10
#define SEIZE           11
#define RELEASE         12
#define ENTER           13
#define LEAVE           14
```

```
#define QUEUE           15
#define DEPART          16
#define GATE            17
#define TEST            18
#define TRANSFER        19
/*
    Additions to commands due to the expansion of the pseudo code
*/
#define EQ              50
#define GT              51
#define LT              52
#define BR              53
#define BE              54
#define BNE             55
#define U               56
#define NU              57
#define SF              58
#define SE              59
#define SNE             60
#define SNF             61
#define NEG             62

/*
    numerical equivalents for GPSS errors
*/
#define NO←END          1
#define REQ←OPER        2
#define LAB←NOT←ALW     3
#define NO←OPER         4
#define SYNTAX          5
#define ILL←OPT         6
#define ILL←LAB         7
#define OPEN←ERR        8
#define FINISH←ERR      9
#define REP←LABEL       10
#define MULT←SIM        11
#define MULT←END        12
#define UN←OPER         13
#define TOO←START       14
#define LAB←REQ         15
#define ILL←NONNUM      16
#define ILL←OPER        17
#define LAB←NO←STAT     18
#define PRE←END         19
#define ILL←FUNC        20
#define MIS←FUNC        21
#define OP←NVLD         22
#define ILL←VAR←DEF     23
#define COMPILER        24
#define UN←LABEL        25
#define PAR10           26
#define STR←NOT←FND     27

/*
    search keys for the symbol table
*/
#define SEARCH          -2

/*
    structure defineitions for the symbol table
*/

struct LAB←                                     /* label attributes */
    {
    int         lab←attribute;                  /* self explanatory */
#       define  LAB←STOR        1
#       define  LAB←LABEL       2
#       define  LAB←VAR         3
#       define  LAB←FUNC        4
#       define  LAB←ASSIGN      5
#       define  SNA←PARAM       6
#       define  SNA←RANDOM      7
#       define  TEMP←VAR        8
#       define  CONSTANT        9
    int         leng←info;                      /* length of storage,... */
    int         *pnt←info;                      /* where information is located */
```

```
    int         func-spec;                  /* type of function - numeric of D for discrete */
    double      *flo-num;                   /* floating point random number data */
    };

typedef struct LAB-     LAB;

struct IDENT-                               /* tree structure for the symbol table */
    {
    char        *label;                     /* where text label is stored */
    LAB         *attr-pnt;                  /* pointer to attributes */
    struct IDENT-       *right;             /* right portion of tree */
    struct IDENT-       *left;              /* left portion of tree */
    };

typedef struct IDENT-   SYM-TABLE;

/*
    structure definitions for info gathered by first pass
*/

struct STRT-
    {
    int         sim-time;
    int         print;
#       define  NP      OFF
#       define  PRT     ON
    int         interval-print;
    };

typedef struct STRT-    INFO-STRT;

/*
    definitions for variable parsing
*/
#define ADDITION        '+'
#define SUBTRACTION     '-'
#define MULTIPLICATION  '*'
#define DIVISION        '/'
#define UNARY           '~'

/*
    structure for the pseudo language
*/

struct CODE-
    {
    int         block;
    int         operator;
    SYM-TABLE   *A-operand;
    SYM-TABLE   *B-operand;
    struct CODE *code-link;
    };

typedef struct CODE- CODE;
```

```
/*+
** NAME:
**              simulator.h
**
** SYNOPSIS:
**              #include "[vandellon.thesis]simulator"
**
** PURPOSE:
**              #include information needed to gather simulation statistics.
**
** RETURNS:
**              Not applicable.
**
** MAINTENANCE:
**              29-MAr-84       jvd     created
**
**
-*/

struct block
    {
    int cur-contents;
    int total;
    };

typedef struct block    BLOCK;

struct fac-stats
    {
    struct fac-stats    *link;
    int name;
    int cum-time;
    int total-ent;
    int seize-block;
    int seized;
    };

typedef struct fac-stats        FAC-STATS;

struct que-stats
    {
    struct que-stats *link;
    int name;
    int max-content;
    int cum-time;
    int total-que;
    int total-ent;
    int zero-ent;
    int cur-contents;
    };

typedef struct que-stats        QUE-STATS;

struct fut-events
    {
    struct fut-events   *for-link;
    struct fut-events   *bck-link;
    struct code         *cur-block;
    int                 BDT;
    int                 truth-bit;
    FAC-STATS           *ptr-fac;
    int                 time-sez;
    QUE-STATS           *ptr-que;
    int                 time-que;
    struct stor-stats   *ptr-stor;
    int                 time-stor;
    int                 blk-from;
    int                 param[10];
    };

typedef struct fut-events       FUT-EVENTS;

struct cur-events
    {
    struct cur-events   *for-link;
```

```
    struct cur-events    *bck-link;
    struct code          *cur-block;
    int                  BDT;
    int                  truth-bit;
    FAC-STATS            *ptr-fac;
    int                  time-sez;
    QUE-STATS            *ptr-que;
    int                  time-que;
    struct stor-stats    *ptr-stor;
    int                  time-stor;
    int                  blk-from;
    int                  param[10];
    };

typedef struct cur-events       CUR-EVENTS;

struct storage-stats
    {
    struct storage-stats        *link;
    char        *stor-nam;
    int         capacity;
    int         cum-tm;
    int         cum-contents;
    int         entries;
    int         cur-cont;
    int         max-cont;
    };

typedef struct storage-stats    STOR;
```

```
!
!       phase 1:        PARSING of the GPSS language
!
$link/executable=gpss gpss,gpspass1,gpserr,getword,keyword,label,strlab,-
                          prssim,prsstr,prsend,prssto,prsvar,prsfun,prsgen,-
                          prster,prsadv,prsass,prsgat,prstst,prsblk1,prstra,-
                          prsblk2,prsnum,prsolb,prslab,prsflo,getvar,itoa,-
                          conunary,vaddsub,vmuldiv,prsymtab,
!
!       phase 2:        PSEUDO CODE generation
!
                          gpspass2,crepsd,ps2tst,-
!
!       phase 3:        SIMULATOR
!
                          gpssim,simgen,simrnd,evaopr,evavar,evafun,fndcol,-
                          srtfut,putcur,addcur,remcur,evasto,crstor,fndstr,-
                          fndque,fndfac,-
                          simadv,simass,simbr,simneg,simter,siment,simlev,-
                          simque,simdep,simsez,simrel,simcmp,simfac,simstor,-
                          simbne,-
                          report
```

13. User Manual.

13.1. Introduction.

The following pages describe a subset of the GPSS language. This workable subset consists of

3	control statements
13	block definition statements
3	entity definition statements

and makes use of 4 numerical and computational attributes.

The following pages use the terminology integer, parameter, variable, function, and storage when describing the operand specifications.

All parameters, variables, functions, and storages used in block definition statements are to be of the form:

s*_storage-label_	:	storage specification
fn*_function-label_	:	function specification
v*_variable-label_	:	variable specification
p*_parameter-#_	:	parameter specification

unless otherwise specified in the documentation.

The thesis GPSS language implementation allows for a free-form style of coding; no column restrictions are in place. Also, capital and lower cases letters are allowed. Please note, when a label has been described in either upper or lower case letters, the references to that label, through out the program, must remain consistent.

The format for the GPSS command line is as follows:

$ gpss *filename.extension*

No qualifiers are allowed on the command line in this version of GPSS.

13.1. Control Statements.

Statement:

END

Syntax:

END

Description:

This must be the last statement of the source GPSS program. This statement signals to the first pass of the assembly phase that there is no more input and indicates the start of the second phase.

Examples:

1. SIMULATE

   .

   .

   END
   Last statement in the GPSS program.

Statement:

SIMULATE

Syntax:

SIMULATE

Description:

A SIMULATE statement requests a simulation run. Without a SIMULATE statement, the GPSS model is parsed but not executed.

Examples:

1. SIMULATE
   .
   .
   .
   END
   Execute the model.

Statement:

START

Syntax:

START A, B, C

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A	required	run termination count	integer
B		print suppression indicator	NP - no statistics printed
			NULL - print
C		snap interval print count	integer

Description:

Operand A specifies the number of terminations to be executed before printing the final summary statistics. When the run termination count is decremented to zero or to a negative value, the run terminates. Each time a transaction enters a TERMINATE block, the run termination count is decremented by the TERMINATE statement's A-operand. The B-operand is self-explanatory. Operand C is a count which will print a normal statistical output when decremented to zero or to a negative value by the A-operand.

Examples:

1. START 10
   Execute the model for 10 transaction terminations.

2. START 10, NP
   Execute the model for 10 transaction terminations and *do not* print the statistical results.

3. START 10, , 2
   Execute the model for 10 transaction terminations and print statistical results after each 2 transaction terminations.

Statement:

ADVANCE

Syntax:

ADVANCE A

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A	required	time delay	variable, function, integer, or parameter

Description:

This block is used to delay entering transactions for the amount of time specified by the A-operand. The transaction is not denied entry into this block.

Examples:

1. ADVANCE 1
   Delay the transaction in this block 1 time unit.

2. ADVANCE P*1
   Delay the transaction for the amount of time specified in parameter 1.

Statement:

ASSIGN

Syntax:

ASSIGN A, [sign] B

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A	required	parameter number	integer, parameter, function, or variable
B	required	value to be added to or subtracted from the parameter	integer, parameter, function, or variable
sign	required	addition or subtraction operator	+ or -

Description:

This block enters or modifies the value of a transaction parameter. The value of operand B is preceded by a + or -. The ASSIGN block never refuses entry to a transaction, and the transaction always will proceed to the next sequential block.

Examples:

1. ASSIGN 1,-2
   Decrement the contents of parameter 1 by 2.

2. ASSIGN 1,+P*2
   Add the contents of parameter 2 to the contents of parameter 1.

Statement:

DEPART

Syntax:

DEPART A, B

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A	required	number of the queue	integer, parameter, variable, or function
B		number of units to be removed from the queue; default is 1	integer, parameter, variable, or function

Description:

This block decreases the queue entity contents by the quantity specified by the B-operand. If the B-operand in not specified, a default of 1 is assumed. No transaction is ever refused entry to a DEPART block. A transaction does not have to remove the same number of units which it added to a queue entity via a QUEUE block.

Examples:

1. DEPART 1
   Remove 1 unit from queue 1.

2. DEPART V*VAR1, P*1
   Remove the number of units specified by parameter 1 from the queue whose number is given by the variable VAR1.

Statement:

ENTER

Syntax:

ENTER A, B

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A	required	storage number or name	integer, parameter, label, variable or function
B		number of units of storage; default is 1	integer, parameter, function, or variable

Description:

Operand A specifies the storage to be accessed, and the B-operand specifies the number of the storage units to be occupied. This block does not permit the transaction to enter if the storage units are full or unavailable. A transaction can enter a different amount of storage units than it leaves via the LEAVE block. If the storage capacity is available, the transaction can enter the block, causing the storage amount to be decremented by the value contained in the B-operand. If the B-operand is not specified, the storage amount is decremented by 1.

Examples:

1. ENTER S*STORAGE1
   The storage specified by the storage label, STORAGE1, is decremented by 1.

2. ENTER 3, 2
   Storage number 3 is decremented by 2.

Statement:

GATE

Syntax:

GATE A B, C

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A	required	comparison qualifier	U facility used NU facility not used SF storage full SE storage empty SNF storage not full SNE storage not empty
B	required	entity number or name	storage label, function, variable, integer, or parameter
C		exit when condition not satisfied	label

Description:

The GATE block alters or impedes the flow of transactions through the model. The status of the particular facility or storage area is tested according to the comparison qualifier specified by the A-operand. If the condition is true, the transaction enters the next block. If the condition is false, the transaction transfers to the label specified in the C-operand, if specified. Otherwise, the transaction is delayed from entering into the block until the test becomes true.

Examples:

1. GATE U 1, LABEL1
   Facility 1 is tested to see if it is utilized. If the comparison is true, the transaction continues to the next block. If false, the transaction goes to the program area specified by LABEL1.

2. GATE SF S*STORAGE1, LABEL1
   The storage specified by the name STORAGE1 is tested to see if it is full. If the comparison is true, the transaction continues to the next block. If false, the transaction goes to the program area specified by LABEL1.

Statement:

GENERATE

Syntax:

GENERATE A, B

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A	required	mean integration time	integer, variable, function,
B		time spread	integer, variable, function,

Description:

This block creates transactions to enter the model at the next sequential block. The creation rate of transactions is specified in terms of an "intercreation" time between successive transactions. The time when the transaction is created is in the range of 'A-operand - B-operand' to 'A-operand + B-operand', if the B-operand is a constant; a random number generator is used to develop an equal distribution in the specified range. The time when the transaction is created is in the range of 'A-operand * B-operand' to 'A-operand * B-operand', if the B-operand is a function or variable.

Examples:

1. GENERATE 100, 5
   Generate a transaction in the range of every 95 to 105 time units.

2. GENERATE FN*FUNC2, V*VAR3
   Generate a transaction in the range of every 'FUNC2 * VAR3' to 'FUNC2 * VAR3' time units.

Statement:

LEAVE

Syntax:

LEAVE A, B

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A	required	storage name or number	integer, variable, function, parameter, or label
B		number of units of storage; default is 1	integer, variable, function, or parameter

Description:

The LEAVE block makes available previously occupied contents in a storage entity. This block never refuses entry to a transaction. The storage area specified by the A-operand has the amount of storage specified in the B-operand made available. If the B-operand is not specified, the default is 1 unit of storage.

Examples:

1. LEAVE S*1, 2
   Storage 1 has its storage amount increased by 2.

2. LEAVE S*STORAGE1
   The storage specified by the name of STORAGE1 has its storage amount increased by the default of 1.

Statement:

QUEUE

Syntax:

QUEUE A, B

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A	required	number of the queue	integer, variable, function, or parameter
B		number of units to be added to the queue; default is 1	integer, variable, function, or parameter

Description:

The QUEUE block is provided to select points in the model where queue sizes need to be measured. No transaction is ever refused entry into a QUEUE block. This block adds the number of units, specified by the B-operand, to the contents of the queue, referenced by the A-operand.

Examples:

1. QUEUE 1, 2
   Queue number 1 gets 2 units of the queue used by the incoming transaction.

2. QUEUE 1
   Queue number 1 gets the default of 1 unit of the queue used by the incoming transaction.

Statement:

RELEASE

Syntax:

RELEASE A

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A	required	number of the facility	integer, variable, function, or parameter

Description:

This block releases a facility previously seized via the SEIZE block, therefore making it available for another transaction. The transaction is never refused entry into the RELEASE block. The releasing transaction must be the transaction which had previously seized the facility.

Examples:

1. RELEASE 1
   Release facility 1.

Statement:

SEIZE

Syntax:

SEIZE A

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A	required	number of the facility	integer, variable, function, or parameter

Description:

This block seizes a facility, making the facility unavailable to any other transactions. A transaction *cannot* enter the SEIZE block if the facility is being utilized. Operand A contains the index number of the facility to be seized.

Examples:

1. SEIZE 1
   Seize facility 1.

Statement:

TERMINATE

Syntax:

TERMINATE A

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A		number of units by which the run termination count is reduced	integer

Description:

This block destroys the entering transaction, thus removing the transaction from the model. A TERMINATE block never refuses a transaction entry. The run termination count is decremented by the value specified by the A-operand. If the A-operand is not specified, the transaction is destroyed but the run termination count is not decremented.

Examples:

1. TERMINATE
   The transaction is destroyed but the run termination count is *not* decremented.

2. TERMINATE 10
   The run termination count is decremented by 10 units.

Statement:

TEST

Syntax:

TEST A B, C, D

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
A	required	algebraic comparison	E equal to L - less than G - greater than
B	required	SNA to be compared	parameter, variable, integer, or function
C	required	SNA to be compared to B	parameter, variable, integer, or function
D		label	valid label

Description:

This block controls the flow of transactions by testing the algebraic relation between operand B and operand C. If operand D is not specified, transactions cannot enter the TEST block until the algebraic relation is true. If operand D is specified, transactions are never refused entry into the block. If the relation between the B and C operand is true as specified by the auxiliary operand A, the transaction proceeds to the next block. If the relation is false, the transaction proceeds to the block specified by operand D.

Examples:

1. TEST E P*1, 1
   Delay the transaction until the contents of parameter 1 is equal to 1.

2. TEST L V*ABC, 2, LABEL1
   If variable ABC is less than 2 continue to the next block, else if false go to the block defined by LABEL1.

Statement:

TRANSFER

Syntax:

```
TRANSFER    ,B
```

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
B	required	label	valid label

Description:

This block unconditionally transfers the transaction to the program block specified by the A-operand.

Examples:

1. TRANSFER ,LABEL1
   Transfer the flow of the program to the block specified by the label, LABEL1.

13.3. Entity-Definition Statements.

Statement:

FUNCTION

Syntax:

label FUNCTION A, B
*decimal*1, *integer*1 / *decimal*2, *integer*2 / ... / *decimal*n, *integer*n

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
label	required	function label	valid label
A	required	random number specification	R
B	required	function and number of points	D*n* where *n* is the number points
*decimal*n	required	number in the range 0.0 - 1.0	decimal number in the form 0.n or .n, where *n* can be multiple integers
*integer*n	required	value of the range	integer value

Description:

The FUNCTION card defines a standard numerical attribute, the type of function, and the number of points that make up the function. Operand A for this project is only defined for one random number. Operand B for this project can ***only*** be defined to be a discrete function with number of points n. Each function must have at least two points defined.

Examples:

1. FUNC1 FUNCTION R, D2
   0.1, 10/ 0.55, 2

Statement:

STORAGE

Syntax:

label STORAGE A

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
label	required	storage label	integer or valid label
A	required	number of storage units	integer

Description:

This card defines the capacity of each storage area that is used in the model. The A-operand contains the capacity assigned to the storage area. Only one storage area can be defined by each STORAGE statement.

Examples:

1. STOR1 STORAGE 1
   The storage area STOR1 is defined to have a capacity of 1 unit.

Statement:

VARIABLE

Syntax:

label VARIABLE A

Operands:

Operand	*Required*	*Description of operand*	*Operand Specifications*
label	required	variable label	valid label
A	required	variable definition	integer, function, other previously defined variables, and parameters are legal arithmetic units; +, -, *, and / are legal arithmetic operators

Description:

This statement defines an arithmetic variable. The variable is evaluated from left to right, with * and / given priority over + and -. No parentheses are allowed.

Examples:

1. VAR1 VARIABLE 32 * 9 + P*2
   The variable, VAR1, multiplies 32 by 9 and adds the result to the contents of parameter 2.